ARMA X

Daniel Bellido Chueco

This document presents the development of my Final Year Project, from the initial aims and objectives to the final implementation and testing. The State of the Art Review section provides a comprehensive analysis of existing games and technologies, leading to the requirements specification and development methodology. The Game Design section includes both the conceptual and technical designs. Implementation details are provided in the following section, while Testing and Evaluation assesses the game's performance. The legal, social, security, and ethical issues related to the game are discussed, followed by a critical review and conclusion. Finally, the document ends with a list of references and technical appendices.

SUPERVISOR
Jarek Francik

Faculty of Science, Engineering and Computing

2022/2023

Kingston University London

Contents

ARMA X - Dissertation

1. Introduction

About this project

This project is a third-person stealth game developed with the Unity Engine called Arma X. Set in a dystopian world where the Nazis conquer Europe during the World War II and maintain power in the old continent until nowadays; the game focuses on the mission of a secret agent who must infiltrate a military base taken over by the Nazi regime in Iceland.

The agent's mission is to find out if the Nazis have moved nuclear weapons on the island with the intention of attacking America by surprise. During this process, the agent confronts a high-ranking SS officer and manages to get hold of top-secret documentation that reveals the true intentions of the regime. These documents contain information about an extremely dangerous weapon known as Arma X.

This secret weapon has the ability to emit powerful electromagnetic pulses that disable all types of electrical devices, which represents a security threat against any country since there would be no way to defend against nuclear attacks.

Regarding gameplay mechanics, the game places significant emphasis on stealth-oriented tactics and devising strategies to outmanoeuvre the AI-controlled enemy units that patrol the game's environment. The player is equipped with a diverse skill set, enabling them to effectively traverse through the game's terrain and overcome various physical obstacles such as fences, barriers, windows, and the ability to climb vertical surfaces including walls, pipes, cornices, and railings.

Furthermore, the player has the capacity to navigate the game environment in different positions, specifically standing, crouching, and crawling, allowing for stealthy infiltration through ventilation shafts. Noise manipulation constitutes a pivotal aspect of gameplay, enabling the player to distract AI through sound-generating actions such as shelling, stone-throwing, or bottle-throwing.

In the event of enemy detection, the player possesses an arsenal of weapons to defend themselves against hostiles, comprising an array of lethal arms including assault rifles, shotguns, sniper rifles, grenades, and rocket launchers. Notably, a silenced pistol constitutes the player's principal tool for stealth eliminations, while non-lethal takedowns may be performed through the execution of enemies from behind when they are distracted. The ability to conceal the corpses of adversaries, as well as the player themselves, is possible through a variety of means including hiding in containers, lockers, and boxes, thereby evading detection and confusing enemies.

Moreover, the game includes the possibility of implementing vehicular transport options, such as off-road cars, trucks, and tanks, to expedite travel and augment the gaming experience.

The gameplay mechanics of the game's primary character are intricate and expansive, necessitating a considerable time investment to master the complex and diverse control mechanisms.

Aims and Objectives

The aim of this project is to create a stealth game that draws inspiration from popular titles such as Metal Gear, Splinter Cell, Wolfenstein, and Assassin's Creed. One of the primary objectives of the project is to develop gameplay mechanics that enable players to experience the role of a spy, complete with various abilities that can be employed to outsmart enemy AI. Accordingly, the project places a significant emphasis on player control, incorporating a parkour system that allows for the traversal of obstacles of varying heights and triggering distinct animations. Furthermore, a climbing system has been implemented to provide players with the ability to climb higher walls, contributing to a more diverse range of gameplay mechanics.

Another fundamental pilar of this project is the creation of a realistic gaming experience through the development of artificial intelligence (AI) that can respond appropriately to specific situations, presenting a challenge for players. To achieve this aim, the AI has been divided into two distinct phases: basic AI and advanced AI. The development of the AI system is a critical component of the project, as it will allow for more immersive gameplay, enhancing the realism of the game world and elevating the player's experience.

In addition to the primary objectives of this project, there is a crucial need to introduce action into the game at specific moments to prevent it from becoming monotonous for the player. To achieve this goal, a diverse set of weaponry will be developed, enabling the player to engage in direct combat with enemies, as well as providing the ability to destroy certain stage elements when the player seeks to exit stealth mode. This approach will enhance the overall gaming experience, adding an additional layer of gameplay mechanics and increasing the level of excitement for players.

The objective of the level design is to ensure coherence with the game's narrative by implementing a quest line that is based on stealthily exploring the area and advancing towards various objectives that are updated upon completion. The game's linear narrative is supported by cinematic elements that offer both playable and cinematic experiences to the player.

To maintain a balanced difficulty level, various types of enemies will be incorporated, each possessing different weapons, resistance to attacks, and reactions to suit the game's requirements. Simple cameras will monitor certain areas, private soldiers will patrol the base, and elite soldiers or vehicles will appear when the alarm is triggered.

The level design should be expansive, offering players side missions and the opportunity to collect items that aid in better comprehending the game's story. Such features allow for player exploration outside of the main quest and provide replayability for players seeking to achieve 100% completion.

There is a plan to develop a Game Manager which will be responsible for managing vital components of the video game, including a quest line that aligns with the storytelling and narrative of the game, in order to enhance its immersive and entertaining aspects. Furthermore, an Audio Manager is also implemented to oversee the management of all in-game sounds and volume.

Summary of the achievements

In relation to the achievements of this project, its most notable achievement pertains primarily to gameplay programming through the implementation of the player controller. Nevertheless, extensive research has also been conducted in other fields, including level design, visual effects, lighting, narrative, and artificial intelligence.

Gameplay Programming

Basic Movement: The fundamental implementation of player movement consists of displacing the player with a Vector3, aided by Cinemachine's FreeLookCamera, which controls camera movement. Furthermore, a BlendTree has been incorporated into the Animator component, allowing for the execution of various movement animations based on input received.

Multiple Stances: Expansion of the basic movement with different stances: Stand, Crouch and Prone. Each stance triggers different animations and modifies the player's speed accordingly. Additionally, the size of the capsule collider is adjusted with respect to each stance, enabling the player to navigate through gaps of different heights such as mid-height gaps or ventilation shafts.

Parkour System: This system allows the player to interact with the environment and perform various actions depending on the type of object in front of him. This detection is based on the combination of two raycasts originating from the player's position that scan the environment and determine what type of object the raycast has hit and how high it is. The different animations to perform are Scritible Objects that contain necessary data that must match the parameters of the object detected by the raycast.

Climbing System: This implementation is an expansion of the Parkour System where the scanner fires multiple raycasts looking for a specific type of object with the ledge tag. When one of these raycasts hits one of these objects it triggers the animation and the climb state, providing a new type of movement within this state and disabling the character's usual state. The player can navigate a network of ledges thanks to the input of a Vector2 and the action button.

Secondary Aim Camera: A secondary camera, which can be activated through input, transitions the perspective from FreeLookCamera to AimCamera, shifting the camera's position over the player's shoulder to offer an improved perspective for aiming. This feature is currently under development, and the necessary adjustments to accommodate this type of camera, which should not rotate around the player, have yet to be implemented. Additionally, Inverse Kinematics must be integrated into the character's rig to facilitate the character's ability to aim freely in the direction of the camera.

Level Design

Terrain: Adapting this tool in an HDRP project has had its difficulties due to certain incompatibilities that some of its features present, such as painting trees or plants. It can be consider an achievement to have made this tool work, as the render pipeline was throwing many errors that made the project unplayable.

Use of Assets: The utilization of assets has played a pivotal role in the recreation of an environment that adheres to the narrative, while also facilitating the creation of a navigable military base with obstacles and other intricacies that serve to enhance the player's immersive experience. This has involved the utilization of both freely available and commercially acquired assets from the Unity Asset Store.

Particle Effects: The present project employs particle effects in the recreation of a rain scenario, wherein numerous droplets are cyclically instantiated. These droplets collide, thereby triggering another particle effect that simulates waves of droplets striking the surface of the ground.

Lighting and Post-processing: Implementation of a pair of post-processing volumes that enhance the lighting and colour of the scene. One of these volumes simulates an underwater environment, while the other creates a darker, fog-laden atmosphere that emulates reduced visibility conditions for both players and enemies.

Additionally, a collection of lights have been incorporated into the scene to produce a more lifelike illumination, which in turn serves to heighten the visual impact of the scene.

Narrative Design*

Story and Plot: Without a doubt, one of the most important sections of this project has been being able to develop a narrative with a hook that draws the player's attention. A narrative design has been carried out, developing the most relevant characters in the story and a script that addresses the first bars of the mission.

Video Introduction: With the help of other programs such as Premiere Pro and After Effects, it has been possible to assemble a video that historically contextualizes the player, placing him in a parallel reality. The video has been built based on real images of the Second World War and an animated map.

Cinematic: The project also presents a cinematic that shows the type of story that is narrated at the beginning of the game. This implementation has been possible thanks to Unity's Timeline tool and Mixamo animations.

Note: The narrative section of this project has been developed for the Game and Media Creation Processes module and has been evaluated by Hope Caton. Therefore, it should not be taken into account for the final grade of the Final Year Project.*

UX Design

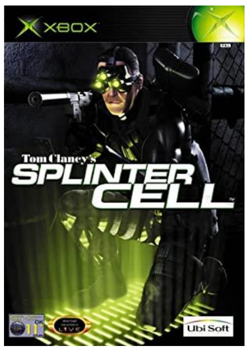
Main Menu: The implementation of a basic and functional Main Menu that provides a minimum of UI that transitions between the introductory video and the cinematic that introduces the player to a new game, in addition to implementing other buttons which some are not functional but will be in future updates.

Loading Screen: The present implementation facilitates the transition from the main menu to any other scene that involves loading game levels. Specifically, the scene loading process is performed asynchronously in order to enable the player to access the scene prior to the full loading of all associated level resources. This design feature enhances the speed and efficiency of transitioning between distinct game scenarios.

2.State of the Art Review

Review of similar games

This project is mostly inspired by many games belonging to the stealth genre but if any have to be mentioned some, those would undoubtedly be sagas like Metal Gear, Splinter Cell and Wolfenstein.



Metal Gear V: Ground Zeros is a prologue to Metal Gear V: The Phantom Pain, or what has been a technical demo of The Phantom Pain. In Ground Zeros there is a relatively small map depicting an American military base where Snake has to locate and rescue two prisoners held at the said base. The control of the main character is simply smooth and satisfying, offering several actions and interactions with the environment that makes the game feel alive. Being a prologue, the narrative is simple but sufficient but what make this game really fun are the fluidity of the protagonist's control and the Artificial Intelligence of the enemies that manage to convey that touch of realism that makes the player feel the adrenaline of actually infiltrating a military base. Being a technical demo, the story is short but very replayable, since it has different missions to choose from that take place in the same base, which lengthens the player's entertainment. This project is based on the proposal of Ground Zeroes, which will try to replicate the control of the main character, advanced AI and replayability in the same scenario.

On the other hand, we have the Splinter Cell saga that, having similar gameplay, offers a slower and more tactical game system with an arsenal of gadgets more appropriate for that style of game, avoiding more direct action with enemies. What make this second title fun are also the adrenaline, the tension, and the satisfaction that comes from completing the level in total stealth. The project adopts some ideas from Splinter Cell, such as playing with stage lighting to hide the player by destroying light bulbs, using indicators on the HUD to know the visibility status, and using gadgets like lock picks and night vision.

Finally, even though it is not a stealth game and it is played in the first person, the project takes as an example and rescues the theme and dystopian setting of the Wolfenstein saga, where the fun is to give the player the opportunity to fight against the Nazi regime in a parallel world that does not exist.

Useful Game Mechanics

Player Controller

Undoubtedly, level navigation is a crucial aspect of gameplay mechanics across various types of games. Smooth and precise control is particularly essential for Metal Gear and Splinter Cell games. The level navigation mechanism must enable players to move freely and derive satisfaction. In numerous games, players encounter obstacles that impede level navigation by preventing them from climbing or crossing them, which results in frustration. Therefore, it is imperative to offer players the freedom to move around the game environment without such limitations. Arma X aims to provide game mechanics that facilitate movement with a level of freedom that emulates real-world movements.

In this regard, Arma X's movement mechanics bear similarities to those in Metal Gear and Splinter Cell, allowing players to stand, crouch, or go prone. Furthermore, the mechanics enable players to sprint and jump over objects such as fences, windows, or walls, rendering level navigation more fluid.

In addition to basic control and a system that allows for parkour, there is also the ability to climb certain surfaces by creating a network of ledges that guide the player from the base of the surface to the top of it. Thanks to this mechanic it is possible to climb different game objects such as cliffs or buildings that help the player avoid areas full of enemies or simply make use of the mechanics to break the monotony proposed by the level. In any case, it is a mechanic that offers many opportunities and does not necessarily have to be limited to the aforementioned.

Another important aspect related to the Player Controller is the combat system. Being a stealth game, the player should be able to knock out enemies without drawing the attention of all of them. In addition, the player will not only have to knock them out but also hide the body of the enemies so that when an enemy discovers the body of an ally it does not trigger the alarm.

But of course, on some occasions, the player will not be able to respect the principles of stealth and in certain situations, confrontation will be necessary, which will set off all the alarms. For this type of situation, it is essential to equip the player with certain weapons that allow him to fight against all types of enemies, be they soldiers, robots, or heavy vehicles. That is why Arma X requires the implementation of a complete inventory that makes available to the player weapons such as assault rifles, shotguns, grenades, and rocket launchers. To facilitate the combat system, an extra camera is needed to switch to an over-the-shoulder perspective, which alters the fundamental movement mechanics to resemble those of the first-person shooter (FPS) genre, as seen in Metal Gear V. In addition to the arsenal, the inventory must also have certain gadgets such as night vision goggles, lock picks, or first aid kits.

Being a stealth game, some game mechanics are essential in a project of these characteristics. Stealth itself is a game mechanic that relies on avoiding detection by the enemy therefore the character has to navigate the level by hiding behind objects and avoiding making noise in order not to attract the attention of the AI.

In this sense, the player has several stances where they will generate a different sound level when moving through the level, but they can also generate noise to distract the AI and breakthrough thanks to this noise generation method. This noise can be caused by items being thrown from the player's position to the desired location as long as it meets a range of requirements that respect real-world physics. In other words, the player will have the ability to throw objects such as bullet casings, stones, or bottles to generate noise and thus intentionally attract the attention of the AI.

Interaction with the environment

The interaction with the environment is a series of mechanics that add realism and fun to the gameplay. In sagas like Metal Gear, it is possible to interact with certain elements of the environment to sabotage the enemy base, such as turning off an electricity generator or destroying enemy communications equipment which means that in case of alarm, the enemy cannot call for reinforcements or the Simply being able to hide in a locker or container to evade enemies expands the gameplay and enriches the player experience.

Other mechanics such as opening a simple door are necessary to add some difficulty to the game and prevent it from being too easy. In addition, this type of mechanics allows inserting small puzzles or mini-games such as picking the lock as happens in games like Splinter Cell or Skyrim to give another example.

Another interesting mechanic in Splinter Cell is to create blind light spots if a bulb or light source is shot to prevent the enemy AI from gaining visibility at that spot. This also requires some kind of on-screen indicator that shows the player how visible they are to an enemy so they know if the player is taking cover in a safe zone or otherwise has to move.

In addition to the aforementioned mechanics, Arma X aims to incorporate driving mechanics into the gameplay. Given the size of the game's proposed level, the inclusion of vehicles such as SUVs, trucks, and tanks could enhance the player's ability to traverse the map and expand the combat system. Moreover, the game's narrative demands the availability of a small aircraft, housed in a hangar, for the player to use in escaping the island upon completion of the game's objectives. This culminating episode, involving a dramatic escape by plane, would provide a high-stakes action sequence for the player to experience.

Artificial Intelligence

Finally, one of the most important pillars, not only of the stealth genre but of video games in general: Artificial intelligence.

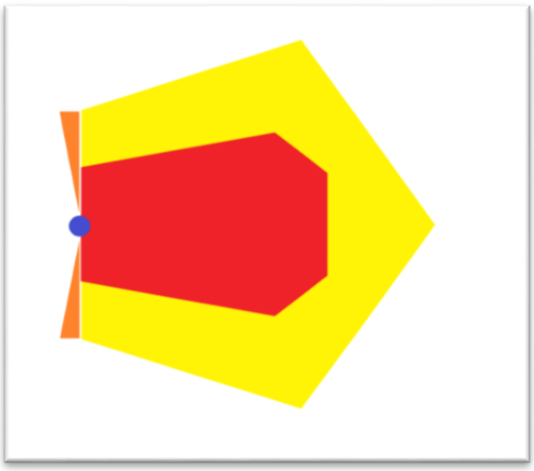
While it is true that a simple Finite State Machine could meet the basic needs of this project, the truth is that titles like Metal Gear V have a much more complex and dynamic AI. It is a field that needs a lot of research and practice to achieve complex behaviours that respond efficiently.

For example, in games like Metal Gear V the AI is so dynamic that its efficiency is affected even by the weather. The level may have a clear day and offer a lot of visibility or there may be a sandstorm that makes visibility difficult and offers small periods where the player can go unnoticed thanks to the complexity of these AIs. These two scenarios where the weather changes dynamically can happen in the same game and the AI has to adapt to any changes that occur in the level.

Another example of how the AI adapts to changes dynamically in Splinter Cell is when the player destroys a light bulb in the environment and reduces the visibility of the AI creating a blind spot where they are safe from the enemy's field of vision.

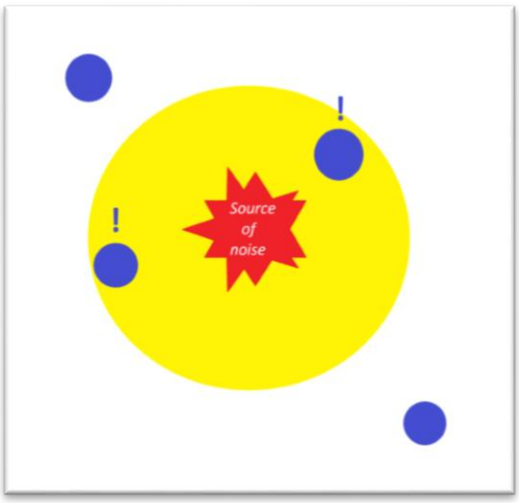
Regarding the AI of this project, the idea is intended to be ambitious and take it as far as possible. The enemy Artificial Intelligence prototype is based on a state machine that will control a series of 12 states or more that could be added in more advanced stages of development. This AI is designed to endow guards with basic senses such as sight and hearing but also some reasoning ability in certain situations.

Sight: The AI field of view will consist of two areas with different magnitudes that will determine if the player has been detected or not. The larger magnitude cone will alert the AI that something unusual is in its field of view so it will decide to approach in (SEARCH) to investigate the location of the object. The smaller magnitude cone will determine what the AI can see clearly and detect the threat directly, so it will trigger directly to the ATTACK state. In addition, two small areas will also be added behind the enemies so that they can notice unusual activity over their shoulder and add a touch of realism (SEARCH).

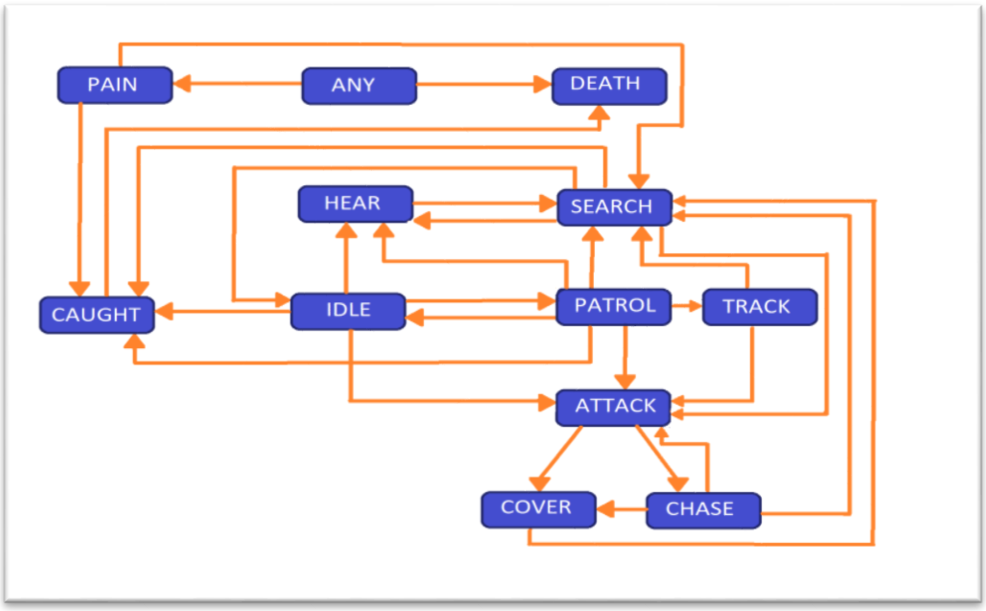


3 fields of view: Area 1(Red), Area 2(Yellow), Area 3(Orange)

Hearing: In order to provide the AI auditory awareness, objects that produce noise will cast a radius, of different sizes depending on the object, which will trigger the AI's reaction when the radius collides with enemies. The radius must pass a vector3 to the AI and this will activate the state SEARCH that will apply the A* algorithm to investigate the incident using the shortest path.



Finite State Machine:



State	Execution	Transition to	Triggered by
IDLE	The IDLE state will simply run an animation of the enemy standing on the spot with a slight body movement to give the impression that they are alive. After approximately 10 seconds, the IDLE state will transition by default to the PATROL state if this is not interrupted by any other event.	PATROL: After approximately 10 seconds, the IDLE state will transition by default to the PATROL state if this is not interrupted by any other event.	PATROL: After patrolling for about 120 seconds, the enemy transitions from PATROL state to IDLE state by default. This creates a loop where the enemy goes from IDLE to PATROL and vice versa automatically by default if no other event triggers another situation.
		ATTACK: If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.	
		SEARCH: If the Player enters Area 2/3 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the SEARCH state and receives a Vector3 location.	SEARCH: After reaching the received location, if there have been no other events so far, the enemy returns to the IDLE state after about 15 seconds.

		HEAR: If the spherical collider produced by an object collides with an enemy, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state.	
		CAUGHT: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.	
PATROL	The PATROL state makes the enemy travel through various waypoints scattered throughout the level for 120 seconds, once the time is up and without any other event having interfered with the routine, the AI transitions to the IDLE state for 10 seconds and then returns to the PATROL state.	IDLE: After 120 seconds, the PATROL state will transition by default to the IDLE state if this is not interrupted by any other event.	IDLE: After approximately 10 seconds, the IDLE state will transition by default to the PATROL state if this is not interrupted by any other event.
		SEARCH: If the Player enters Area 2/3 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the SEARCH state and receives a Vector3 location.	
		HEAR: If the spherical collider produced by an object collides with an enemy, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state.	
		ATTACK: If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.	
		CAUGHT: If the Player collides	

		with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.	
HEAR	The HEAR state plays an animation where the enemy stops confused and receives a Vector3 coordinate and then goes into the SEARCH state.	SEARCH: If the enemy receives the Vector3 coordinate, it starts its journey towards the location.	IDLE: If the spherical collider produced by an object collides with an enemy and the line of sight is FALSE, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state.
			PATROL: If the spherical collider produced by an object collides with an enemy and the line of sight is FALSE, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state.
		CAUGHT: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.	SEARCH: If the spherical collider produced by an object collides with an enemy and the line of sight is FALSE, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state again.
			TRACK: NOT DEFINED YET
SEARCH	The search state plays an animation of the alert enemy moving towards a specific location. This state is triggered by visual activation when the player enters the area of vision of greater magnitude and the line of sight is TRUE, receiving the last Vector 3 coordinates from the player	IDLE: After some seconds, if the enemy AI does not receive any other input the AI goes back to the IDLE state after reaching the Vector3 location.	HEAR: If the spherical collider produced by an object collides with an enemy and the line of sight is FALSE, the AI goes to the HEAR state,

	and applying the A* algorithm to arrive at the location of the most efficient way possible. This state can also be triggered by the sound produced by objects through colliders that will transfer the Vector3 to the enemy so that in the same way they apply the A* algorithm to investigate the area as soon as possible.		receives a Vector3 location, and then to the SEARCH state again.
		HEAR: If the spherical collider produced by an object collides with an enemy, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state again.	PAIN: If the enemy gets shot but not killed, the pain state plays an animation and goes to the search state.
			CHASE: If the enemy sight of line is FALSE after chasing the player for a while, it goes back to the SEARCH state and last player location.
		ATTACK: If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.	COVER: After reloading the weapon or recover some life, the enemy enters to the SEARCH state and goes to the last player location.
		CAUGHT: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.	TRACK: If enemy finds and unusual object on the way, it follows the track defined by the object.
TRACK	The track state consists of making the enemy perceive some foreign object or change in the environment that was not there before. For example, footprints, blood on the ground, or other objects can cause confusion and alert the enemy.	HEAR: If the spherical collider produced by an object collides with an enemy and the line of sight is FALSE, the AI goes to the HEAR state, receives a Vector3 location, and then to the SEARCH state again.	PATROL: If enemy finds and unusual object on the way, it follows the track defined by the object.

		ATTACK: If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.	
ATTACK	If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy attacks the player until his magazine runs out of bullets, if his life is in danger or if he loses sight of the player.	COVER: The enemy takes cover to reload his weapon. The enemy takes cover if his life is under 30%	IDLE: if the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.
			PATROL: if the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.
		CHASE: The enemy chases the player if the player moves away from the Area 1 of sight and the line of sight is TRUE .	TRACK: if the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.
			SEARCH: if the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.
			CHASE: if the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.
COVER	The enemy takes cover to reload his weapon or if his life is under 30%. The enemy moves to the closest safe area to take cover and move to the next state.	ATTACK: If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.	ATTACK: If the Player enters Area 1 of sight and the line of sight or ray cast is TRUE, the enemy transitions to the ATTACK state.

CHASE	The enemy runs after the player until he is in range to attack	ATTACK: When the enemy is close enough to the player, they attack	ATTACK: If the player moves away from attack range, the enemy chases the player.
		SEARCH: If the enemy sight of line is FALSE it goes to the search state.	
		COVER: If the enemy runs out of ammo or their life is under 30%, they take cover.	
CAUGHT	If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.	DEATH: After player executes enemy, this one enters to the DEATH state.	IDLE: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.
			PATROL: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.
			SEARCH: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.
			PAIN: If the Player collides with the enemy from behind and the line of sight is FALSE, the player immobilizes the enemy leaving them defenceless.

PAIN	If the enemy receives damage from the player, it triggers this state for a short period leaving the enemy vulnerable.	SEARCH After the PAIN animation is played, enemy SEARCH.	ANY
DEATH	Plays death animation and stays in this state permanently.	NONE	ANY

Technologies, algorithms and techniques

The project must work with multiple technologies and disciplines to be able to carry out the work.

First of all, one of the most important implementations is to control the player and offer a set of animated movements that allow actions such as jumping over a container, window, or other object in the scene. To do this, the character will need to know what objects are in front of him, so it is necessary to implement a scanner that allows the character to detect the different types of objects that are scattered throughout the level. To make this scanner work, it is necessary to resort to Raycast technology which is in charge of colliding with the scenario and sending the received data to the player to process them as best suited.

In order for the character animations to fit the object's parameters, it is necessary to use the target-matching technique.

Target matching in Unity is a method that allows developers to align and match the position, rotation, and scale of one game object to another. It simplifies the process of positioning objects in a scene and helps to create more realistic and immersive environments. By using target matching, developers can ensure that objects move and behave in a way that is consistent with the game world, making it easier to create engaging and believable experiences for players.

Another technique that requires consideration in this project is Inverse Kinematics, which ensures that animations conform to the gameplay requirements. The implementation of this technique is necessary when transitioning the player from the free camera to the secondary aiming camera, which must move parallel to the character's spine as the player directs their aim upwards, downwards, or to the sides. The use of Inverse Kinematics ensures that the movements of the character's limbs and body are realistic and align with the gameplay's needs.

But of course one of the subjects that deserve special attention is Artificial Intelligence. Being a game based on stealth, this topic needs a lot of research time.

The most common approach is to create a Finite State Machine that contains all the necessary behaviours of the enemy AI. However, sticking to just this technique can result in predictable and somewhat dull AI. For this reason, it is important to try to combine the state machine with some other technique that improves the behaviour of the enemies and produces more dynamic behaviours. In this sense, there are several techniques to explore, such as Unity Machine Learning Agents and Goal Oriented Action Planning.

The Goal Oriented Action Planning (GOAP) is a decision-making AI system commonly used in game development to create realistic and adaptive behaviours in non-playable characters (NPCs). This technique involves modelling the NPC's goals and available actions, then using an algorithm to determine the best sequence of actions to achieve the goal. By using GOAP, NPCs can react dynamically to changing game conditions and exhibit a wider range of behaviours.

For this, it is necessary to investigate further and take the online course offered by the Professor of Games Development, Artificial Intelligence and Computer Science, Penny de Byl in Holistic3D.

Development Tools

The development tools used in this project include Unity as the game engine, with the project type being HDRP. In addition, ProBuilder, Cinemachine, Timeline, Mecanim, and Animation Rigging are used as engine packages to assist in development. The primary programming language is C#, and the development environment is Windows 10 with Visual Studio 2019. To manage version control, Gitlab is utilized, while Trello is used for project management and bug tracking. These specialized tools help to streamline the development process and improve efficiency.

- Game Engine: Unity 2021.3.20f1
- Project type: HDRP
- Engine Packages:
  - ProBuilder
  - Cinemachine
  - Timeline
  - Animation Rigging
  - Mecanim

- Programming Languages: C#
- Development Environment: Windows 10, Visual Studio 2019
- Specialized Tools: Gitlab for version control, Trello for project management, and progress tracking.

3. Analysis, requirements specification and development methodology

MoSCoW Analysis

The MoSCoW analysis is a method used to prioritize requirements in software development projects. In the context of game development, this analysis helps to identify the fundamental features of the game that must be implemented and those that are desirable but dispensable for a basic demo. The table provided outlines the requirements categorized into four levels of priority: MUST, SHOULD, COULD, and WON'T.

MUST	SHOULD	COULD	WON'T
Basic Movement	**Climbing System**	**Collectibles**	**Multiplayer**
Multiple Stances	**Combat System**	**Achievement System/PS Trophies**	**Ads**
Parkour System	**Stealth Movement (cover/wall walk)**	**Vehicles**	**Micro-Transactions**
Large level	**Throw Objects**	**Bullet time**	**Mobile port**
Basic AI	**Advanced AI**	**First person perspective**	**Multiple maps**
Silent Gun	**Weapons/Inventory**	**Save/Load Game System**	**Player Customisation**
Health Bar	**Checkpoints**	**Boss Stage**	
Console Controller	**Environment Interaction**	**Secondary Questline**	
A Questline/ objectives	**Cut-scenes**		
Sound	**UX/UI**		

It is evident that the game with the listed characteristics has the potential to become highly complex, necessitating the implementation of unforeseen features. However, the fundamental requirements can be seen in the MUST column, which outlines the essential player control features, parkour system, console controller, enemy AI features, silent gun, one big level, main menu, and a set of specific linear missions.

The SHOULD column contains desirable features such as cover, obstacle jumping, environment interaction, enemy AI features, enemy sight areas, and the radar on the screen. These elements are dispensable for a basic demo but expected to be included in the final project.

The COULD column consists of additional features that are not fundamental but could enhance the gameplay experience. Some of the elements in the COULD column could be moved to the SHOULD column, including the boss stage, player customization, PlayStation trophies/achievements, collectibles/diary that explains the historical context, and level repetition with different missions. The game does not contemplate any type of multiplayer functionality, micro transactions, or a possible port to mobile devices beyond Nintendo Switch, indicating a focus on the classic single-player story mode.

Overall, the MoSCoW analysis provides a structured approach to prioritize the requirements for game development. By focusing on the essential features while also considering desirable and additional features, game developers can design a game that meets the expectations of players while also ensuring that the development process remains manageable and efficient.

Requirements specification

MUST

Basic Movement: Essential for the player to navigate the level and reach the final goal. This movement is based on moving in any direction on a 3D plane that allows character rotation and third-person camera control.

Multiple Stances: Expansion of the basic movement with different postures: Stand, Crouch, and Prone. Each stance triggers different animations and modifies the player's speed accordingly. Additionally, the size of the pod collider adjusts with respect to each stance, allowing the player to navigate through spaces of different heights, such as mid-height spaces or ventilation shafts.

Parkour System: This system helps the player to navigate the level with much more fluidity and the ability to go over fences, walls, windows, and other obstacles that have a relatively low elevation, such as boxes, containers, or large steps.

Large level: Regardless of size, a level is necessary and preferably one that offers variety and is visually appealing. It is true that for a prototype it is enough to use blocks but for the final product, a complete level is necessary.

Basic AI: A stealth game without AI cannot be a stealth game. Basic AI is absolutely necessary for a prototype of this nature and you should at least have a simple Finite State Machine that handles basic behaviours.

Silent Gun: Basic weapon to be able to get rid of enemies without triggering alarms and having a minimum Combat System.

Health Bar: Life system that makes the player vulnerable and subject to the Game Over condition. Without this implementation, the game becomes extremely easy and boring.

Console Controller: The implementation of a gamepad is necessary since the preference of the players for this type of game requires this type of control and also because the consoles are a fundamental part of the market.

Questline/objectives: Questlines are important in video games because they provide a coherent narrative structure, a sense of progress and achievement for the player, rewards, and motivation to keep playing.

Sound: Sound is a key element in any game. It can add depth, provide feedback, add emotion, enhance the narrative, and provide visual clues. Without sound, the player's experience in a game would not be as complete or satisfying.

SHOULD

Climbing System: This system is an expansion of the mechanics of movement and interaction with the environment. It enriches the gameplay and the player experience by incorporating a complex and varied movement set.

Combat System: A combat system cannot be limited simply to the use of a pistol. This system should offer the player a wide range of options, from stealthily knocking out enemies to confronting them more violently with weapons such as assault rifles, shotguns, grenades, and other weapons. This system must offer a change of perspective when aiming the weapon to facilitate aiming and to be able to shoot the enemy more comfortably.

Stealth Wall Movement: This gameplay mechanic enhances the character's mobility, while also serving as a valuable strategy for seeking cover from enemies both in stealth and combat scenarios.

Throw Objects: The primary objective of this gameplay mechanic is to produce sounds that divert the attention of the AI. Through this mechanic, players can progress through the level by utilizing objects like bottles or bullet casings to lure enemy soldiers away from their observation posts. Furthermore, this mechanic can also be leveraged in the game's Combat System, whereby grenades can be launched using this mechanism.

Advanced AI: Since the main character has the ability to throw objects to distract the AI, it is necessary for the AI to respond to these kinds of events effectively. Therefore, it is necessary to implement other Artificial Intelligence techniques that go beyond the simple states that a State Machine can offer. Another technique that can be combined with the State Machine is the Goal Oriented Action Plan (GOAP) and this will improve the behavior of NPCs in a more dynamic way. Furthermore, NPCs must react accordingly to the environment and perform actions such as jumping over fences or taking cover when the player shoots at them.

Weapons/Inventory: The Combat System requires a diverse arsenal to confront the enemies, it is necessary to implement an inventory that is capable of managing all these items and that the player can select the required item in a given situation. The inventory is not only limited to storing weapons but will also be used to manage other gadgets and utensils such as lock picks, night vision goggles or first-aid kits.

Checkpoints: Checkpoints are an important game design feature that provides players with a sense of progress and accomplishment, balances the challenge of a game, and enhances the storytelling experience. They are a useful tool to create enjoyable games, while also making sure that players are not discouraged by frequent setbacks or difficulties. Incorporating checkpoints into the game design can improve player engagement and satisfaction, ultimately leading to a more successful game.

Environment Interaction: The implementation of interaction with the environment in a game is a complex and extensive requirement, as it can be closely tied to other requirements that may not be fully realized until they are implemented. For instance, shooting a light bulb to create a blind spot and evade detection by the AI has a dependency on advanced AI. Without advanced AI, destroying the bulb to alter the scene's lighting will not have the desired effect. Conversely, interactions with the environment can be straightforward, such as opening a door, triggering an animation, or exploding a barrel.

This project acknowledges the inclusion of interactions with the environment, such as interacting with containers or lockers to hide from enemies, or utilizing objects like computers and buttons to solve puzzles. While the implementation of these interactions may vary in complexity, they are all crucial elements in enhancing the player's experience and immersing them in the game's environment.

Cut-scenes: Cinematics are an important tool that enhances the storytelling of the game. They can provide players with a deeper understanding of the game world and its characters, and help create a more immersive and engaging experience.

UX/UI: The implementation of the UI is not essential in the game, but it helps to communicate the state of the game through indicators such as the life bar, the ammo available if it has been discovered by an enemy, or it can even notify the player. player whether an action is possible when near an object. An example would be to communicate to the player when he can make use of the Parkour or the Climbing System.

What is more important is that there is a transition between the intro video of the game and the game itself. This requires the implementation of a Main Menu where the player can start or resume a previous game thanks to the Save/Load Game System.

COULD

Vehicles: Given the size of the game's proposed level, the inclusion of vehicles such as SUVs, trucks, and tanks could enhance the player's ability to traverse the map and expand the combat system.

Bullet time: This mechanic could be incorporated when the player has been discovered by an enemy. The mechanic itself consists of giving the player a chance to get rid of an enemy before it triggers the alarm.

First person perspective: In addition to adding an over-the-shoulder aiming camera to improve player perspective, a third first-person camera could be implemented to further improve the aiming mechanic, especially when bullet time kicks in upon being spotted by an enemy.

Boss Stage: The project is designed to contain a character identified as a boss but the design does not contemplate that there is a long one-on-one fight with different phases. Although it could have a fight of these characteristics, the boss stage of this project would be like eliminating one more enemy with the accompaniment of some cinematic that helps to distinguish the moment of the game with some relevance.

Secondary Questline: The incorporation of subplots and secondary missions into the game's plot would enrich the gameplay and the narrative experience with the inclusion of extra characters and missions.

Collectibles: Some of these side quests could be to collect collectibles that support the game's narrative by explaining past events during the war. These objects could be letters, photographs, coins, cassette tapes, or other objects that upon examination would tell a piece of the story.

Achievement System: The use of achievements in video games provides several benefits, including enhancing the player's experience by motivating them to explore and complete the game's content, providing a sense of accomplishment, and encouraging replayability. Achievements also promote competition among players and can serve as a means of social interaction and communication within the gaming community.

Development methodology

This project is characterized by an Agile development methodology, which is necessitated by the need for flexibility and an iterative approach that can adapt to changing project needs. The project iterations have been conducted monthly, with periodic reviews of priorities to address evolving requirements and looming deadlines.

Due to the large amount of content and features that must be implemented, the project will be developed using the agile with kanban methodology since it allows different functions to be developed in parallel without the need to have others completely finished. Thus, if there is some kind of stagnation in the development process when it comes to producing a specific implementation, you can start with another task in order to speed up the process and increase productivity by managing the short period of time to develop the prototype. Due to the complexity of the project for a single person, the agile methodology also allows flexibility in certain implementations that were planned but may not be implemented due to lack of time. In the same way, other functionalities can be added in the middle of the development process since the project will be constantly tested and an attempt will be made to balance its needs.

The development progress will be planned in stages where each stage will cover a series of needs based on the MoSCoW analysis. These stages are: **Player Control Stage**, **Level Design 1**, **Environment Interaction Stage**, **Enemy and AI Implementation**, **UI**, **Object Interaction**, **Level Design 2**, **Sound Stage**, **Graphics and optimization Stage**. At this point, and assessing the time remaining until the delivery date, we will try to add **other technical implementations** and improvements.

In the **Player Control Stage**, basic player functions will be implemented to be able to move around the level and perform actions typical of a stealth game such as walking, running, crouching, sneaking, lying down, crawling, aiming with the weapon, shooting and reloading.

In **the first part of the level design**, a suitable environment will be created so that the player's movements are adjusted to the desired gameplay. Objects such as destructible light bulbs, objects to cover the player with, and spaces to hide in will be a priority in this first phase of level development.

The next milestone will be the **Environment interaction** to expand the player's actions in the scene and have a much more complete Player Control. Actions such as taking cover, jumping over obstacles, climbing ladders and walls, grabbing the enemy, executing the enemy, and carrying the enemy's body will be implemented in this stage.

The next objective to tackle is the implementation of **enemies and advanced Artificial Intelligence** which will go through two phases. In the first, enemies and a simple state machine will be implemented that will include basic behaviour such as IDLE, PATROL, ATTACK, CHASE, CAUGHT, and DEATH. In the second phase, more advanced behaviours will be implemented that will add realism to the behaviour of enemies on stage. These advanced behaviours are searching the player, reacting to auditory triggers, analyzing unexpected objects in the scene, and perfecting the sight areas by different levels.

Once the Artificial Intelligence is done, the **User Interface** will be implemented and it will ensure that all the indicators work correctly plus the implementation of **the inventory** system so that the player can quickly select equipment with the controller shortcuts.

The next set of implementations corresponds to **the Object Interaction**, which includes a series of interactions between the player and various objects such as Radio, Camera, Computers, doors and locked doors, containers, lockers, bullet casings, Drone, light bulbs, explosive barrels, and vehicles. Some of these features will have to be partially implemented during the first phase of level design, such as light bulbs and doors. The rest will be implemented progressively depending on the degree of priority that each item had in the game. The least priority but no less important would be the implementation of vehicles, which does not have a relevant impact during the course of the game and its objectives but adds extra fun to the final product.

Object Interaction Priorities:

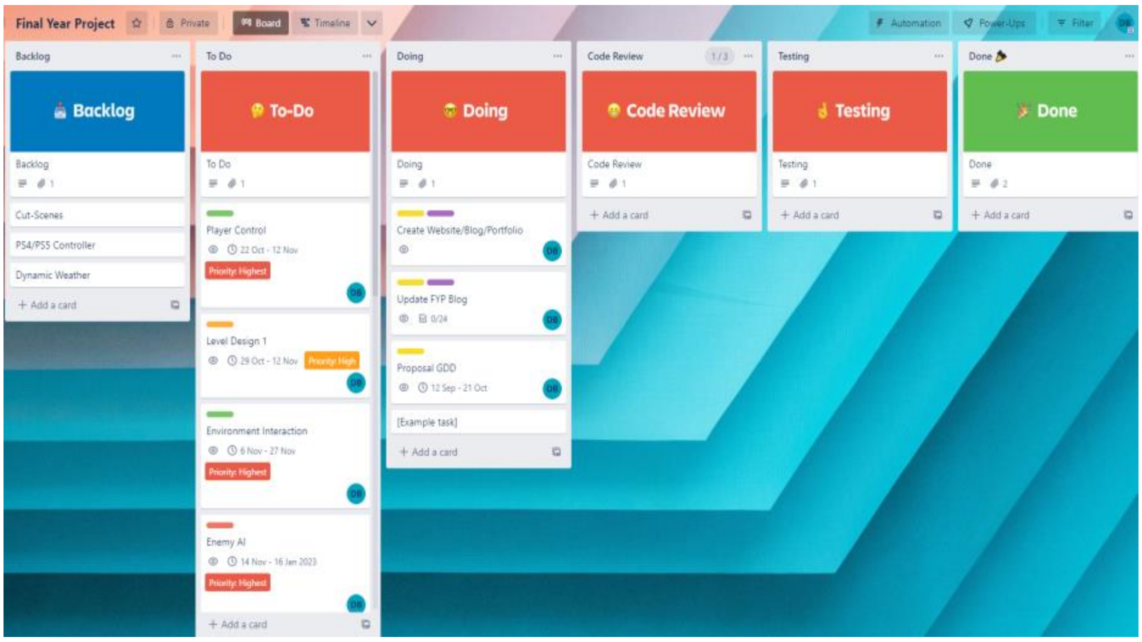
Item	High Priority	Medium Priority	Low Priority
Radio		X	
Camera			X
Computers		X	
Doors	X		
Locked doors	X		
Containers and lockers		X	
Bullet casings	X		
Drone			X
Light Bulbs	X		
Explosive Barrels			X
Vehicles			X

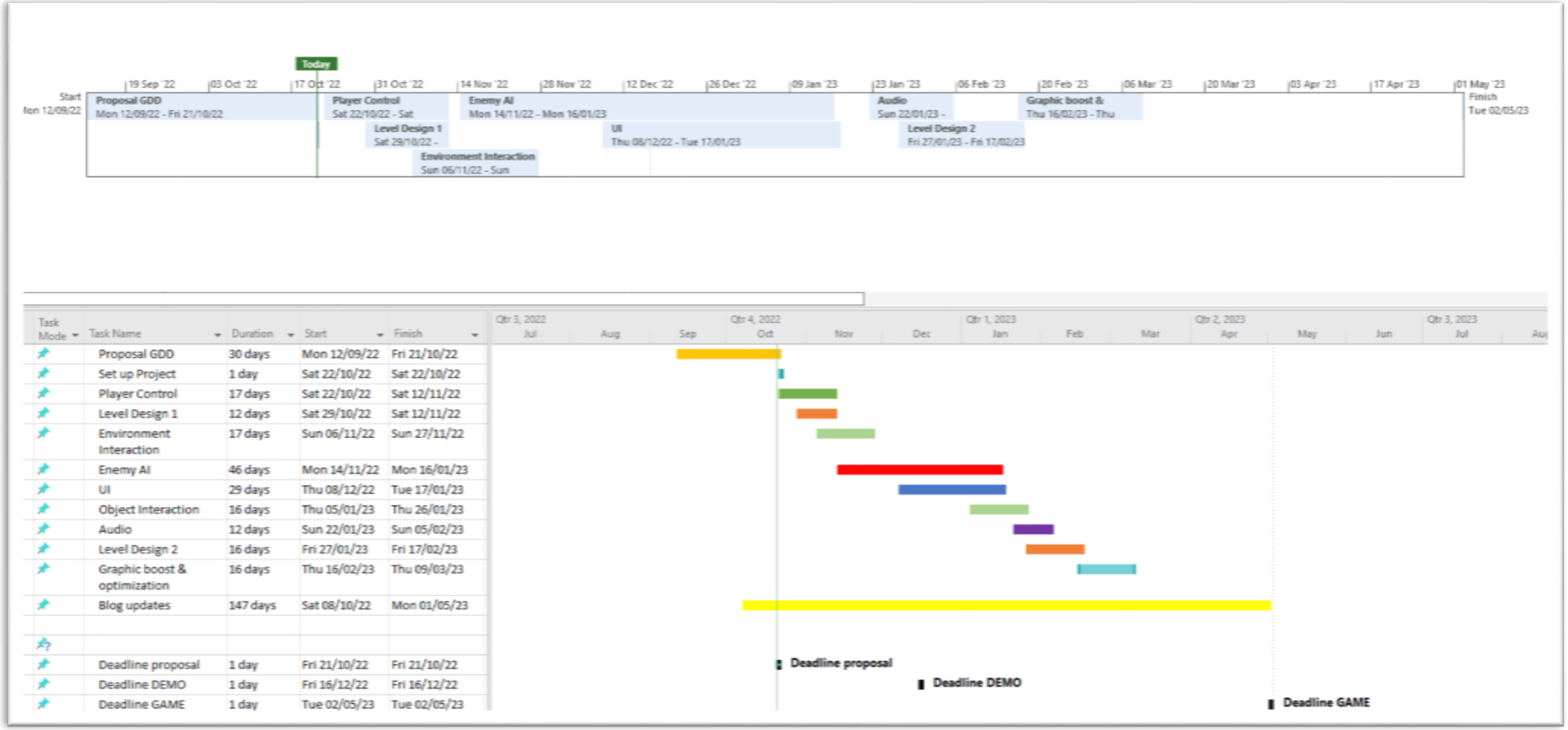
After having implemented the high and medium-priority objects, **the second phase of the level design** will begin to be developed, which will consist of making sense of the game's narrative. Designing missions, puzzles, and a series of tasks so that the player feels that what he is playing has a ludonnarrative harmony. In this phase of the project, we will also try to add cut scenes to reinforce that feeling of living a story but this feature will be considered low-priority until other more needed features have been implemented.

Sound is undoubtedly one of the most important features to provide feedback to the player. It's also an important part of the experience as it accompanies the player's emotions and somehow sub-communicates part of the story. The game will necessarily have music and sound effects as it is an important element in the Artificial Intelligence design, but it will not be fully implemented until more advanced stages of production.

After the implementation of the audio, we will try to **improve the game visually** with the application of shaders and particle effects such as explosions, rain, smoke, and other effects such as night vision goggles to bring the environment where the story takes place to live. It will also try to optimize the game so that the program runs at a stable frame rate and memory space is optimized.

The project management will be supported by a Kanban board that trello.com offers for free. The different blocks of tasks mentioned above are organized in order of priority and with a start date and an approximate date to finish said block. The Trello application itself automatically generates a Gantt chart after having specified these dates as can be seen in the images. Check the link for more detailed view: https://trello.com/b/iss6sd77/final-year-project/timeline





Task Mode	Task Name	Duration	Start	Finish
	Proposal GDD	30 days	Mon 12/09/22	Fri 21/10/22
	Set up Project	1 day	Sat 22/10/22	Sat 22/10/22
	Player Control	17 days	Sat 22/10/22	Sat 12/11/22
	Level Design 1	12 days	Sat 29/10/22	Sat 12/11/22
	Environment Interaction	17 days	Sun 06/11/22	Sun 27/11/22
	Enemy AI	46 days	Mon 14/11/22	Mon 16/01/23
	UI	29 days	Thu 08/12/22	Tue 17/01/23
	Object Interaction	16 days	Thu 05/01/23	Thu 26/01/23
	Audio	12 days	Sun 22/01/23	Sun 05/02/23
	Level Design 2	16 days	Fri 27/01/23	Fri 17/02/23
	Graphic boost & optimization	16 days	Thu 16/02/23	Thu 09/03/23
	Blog updates	147 days	Sat 08/10/22	Mon 01/05/23
	Deadline proposal	1 day	Fri 21/10/22	Fri 21/10/22
	Deadline DEMO	1 day	Fri 16/12/22	Fri 16/12/22
	Deadline GAME	1 day	Tue 02/05/23	Tue 02/05/23

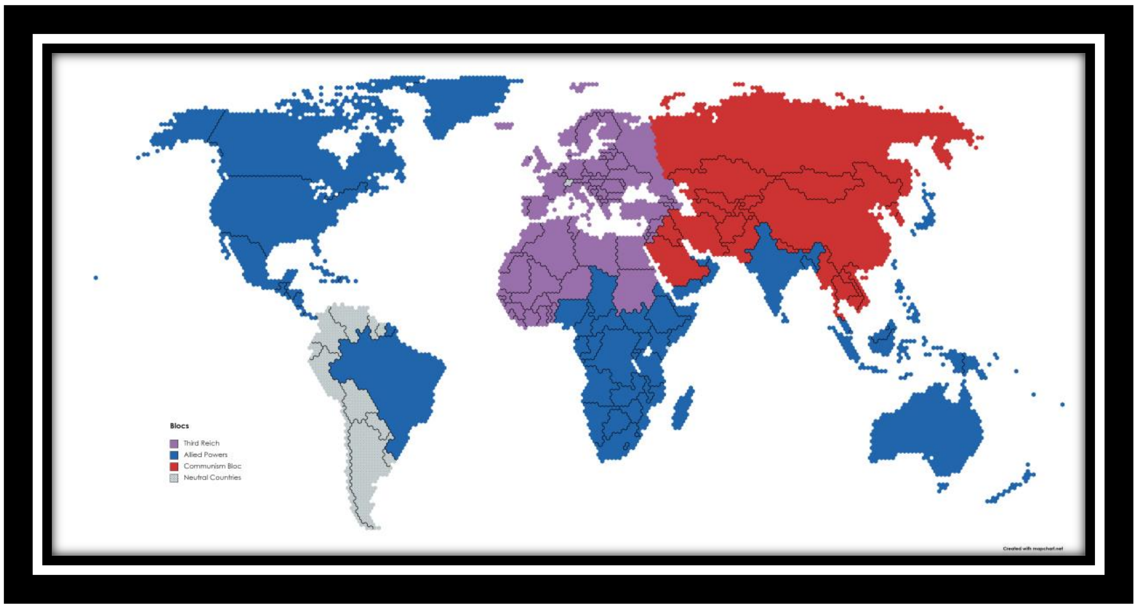
4. Game Design

Conceptual Design

Theme

The game is set in a dystopian alternative world where Nazi Germany emerges victorious in the D-Day Battle of World War II. They successfully conquer Europe, the United Kingdom, and a portion of the Soviet Union and after years of warfare and casualties, the involved nations come to a truce where the Third Reich, alongside its puppet governments, establishes itself as a dominant European empire.

Despite agreeing to a ceasefire with Americans, strained relations persist between the United States, the Soviet Union, China, and Nazi Europe. This cold war atmosphere is exacerbated by the presence of nuclear weapons worldwide.



Storyline

The story follows a special SIS agent who is tasked with infiltrating a Nazi military base in Iceland to verify if the Nazis have deployed nuclear weapons, obtain secret documentation, and eliminate the SS officer in charge. In the first act, the agent successfully infiltrates the base despite facing numerous challenges and obstacles. The second act focuses on the agent's attempt to find the Nukes, eliminate the SS commander, and retrieve the secret documentation, which is filled with dangerous encounters and unexpected twists. The final act reveals that the Nazis not only have deployed nuclear weapons but also possess a secret weapon known as the Arma X, capable of emitting a powerful electromagnetic pulse that disables all electronic devices within a given radius. The agent successfully completes the mission, but the future remains uncertain as the world tensions continue to escalate.

QUESTLINE
Reach the perimeter of the base undetected
Find a way to penetrate the base
Locate the SS commander's office
Find the nuclear weapons
Eliminate the SS commander
Retrieve the secret documentation
Escape from the enemy base

Game Mechanics

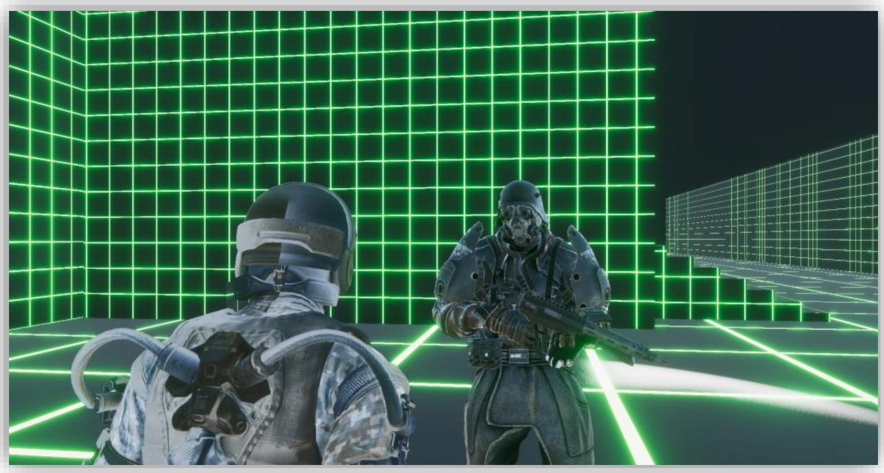
The game prioritizes stealth-based tactics and strategy to navigate the game's environment, which is patrolled by AI-controlled enemies. The player possesses a versatile skill set that enables them to move through the terrain and overcome physical obstacles, including climbing walls, pipes, and railings. The player can also navigate the environment in different positions such as standing, crouching, and crawling, which allows for stealthy infiltration. The game emphasizes noise manipulation as a vital component of gameplay, which allows the player to distract enemies with sound-generating actions like shelling, stone-throwing, or bottle-throwing.

If detected by enemies, the player has access to an array of lethal weapons, including assault rifles, shotguns, sniper rifles, grenades, and rocket launchers. The player's primary tool for stealth eliminations is a silenced pistol, and non-lethal takedowns are possible by executing enemies from behind when they are distracted. The game also allows players to conceal corpses and themselves in containers, lockers, and boxes to evade detection and disorient enemies.

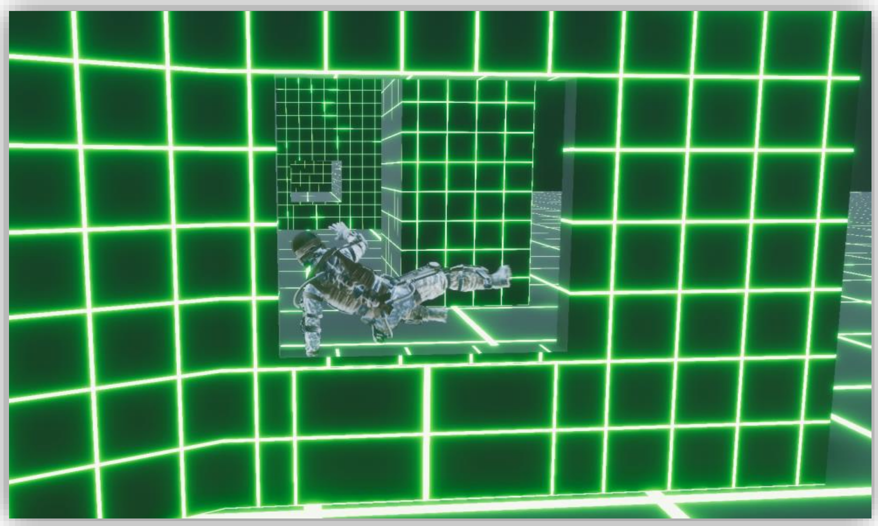
In addition, the game offers vehicular transportation options, such as off-road cars, trucks, and tanks, to enhance the gameplay experience and expedite travel.

Appearance

In terms of appearance, the game features highly-detailed and immersive environments, which are rendered in realistic and visually-striking graphics. The game's setting is often dark and gritty, with a focus on industrial and military themes, creating a sense of tension and danger for the player. The player character is typically equipped with gear and clothing appropriate for stealth and combat, including camouflage and body armour. The game also includes a variety of enemies, each with distinct appearances and behaviours, adding to the game overall immersion and realism. Additionally, the game features cinematic cutscenes and scripted events, further enhancing the visual and narrative aspects of the gameplay experience.



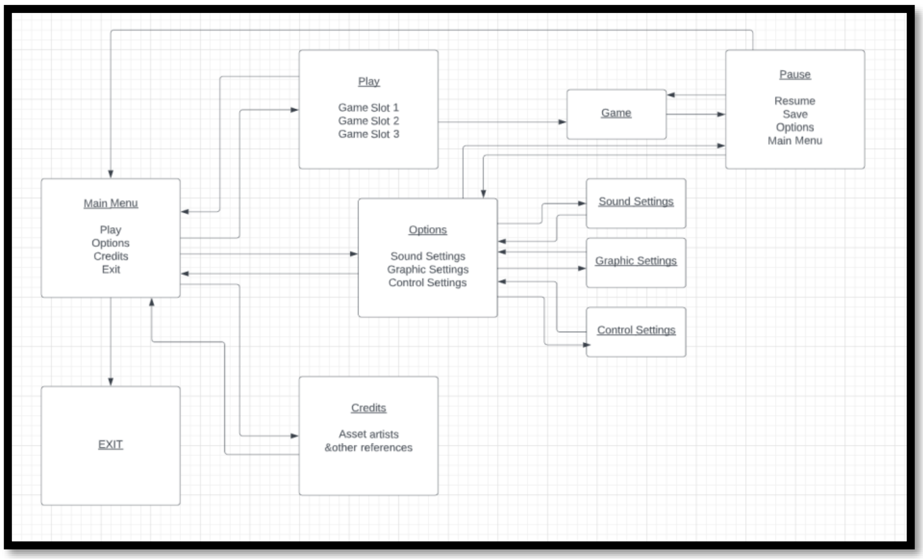
Main character in front of an enemy soldier (Training room)



Jumping out a window. Parkour mechanic.

GUI

The graphical user interface (GUI) is an essential component of video games that provides players with a way to interact with the game's world and mechanics through visual and interactive elements. It includes menus, buttons, icons, and other elements that facilitate navigation, control, and customization. The GUI is crucial for player engagement and immersion, as it allows players to understand the game's mechanics and objectives, access game options and settings, and monitor their progress. A well-designed GUI can enhance the player's experience, make the game more accessible, and improve usability.

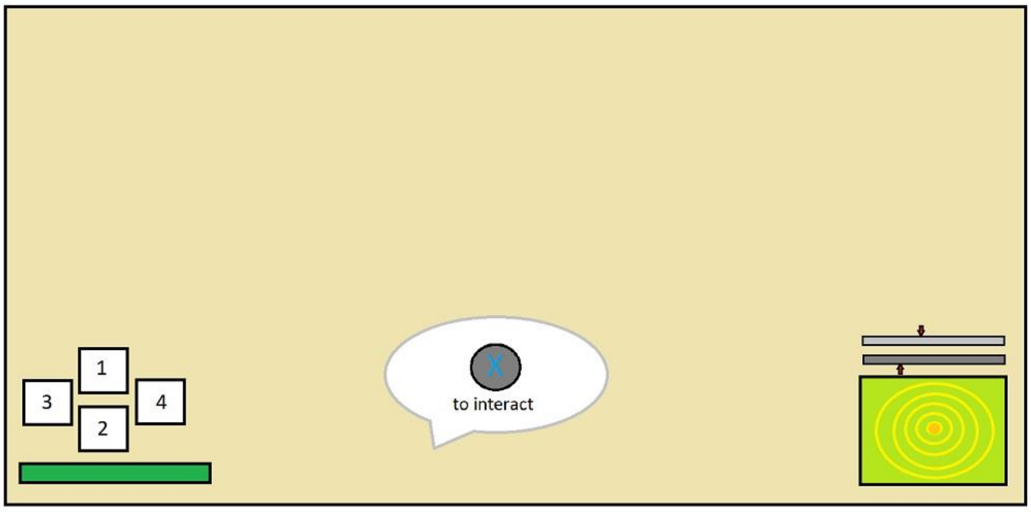


App navigation

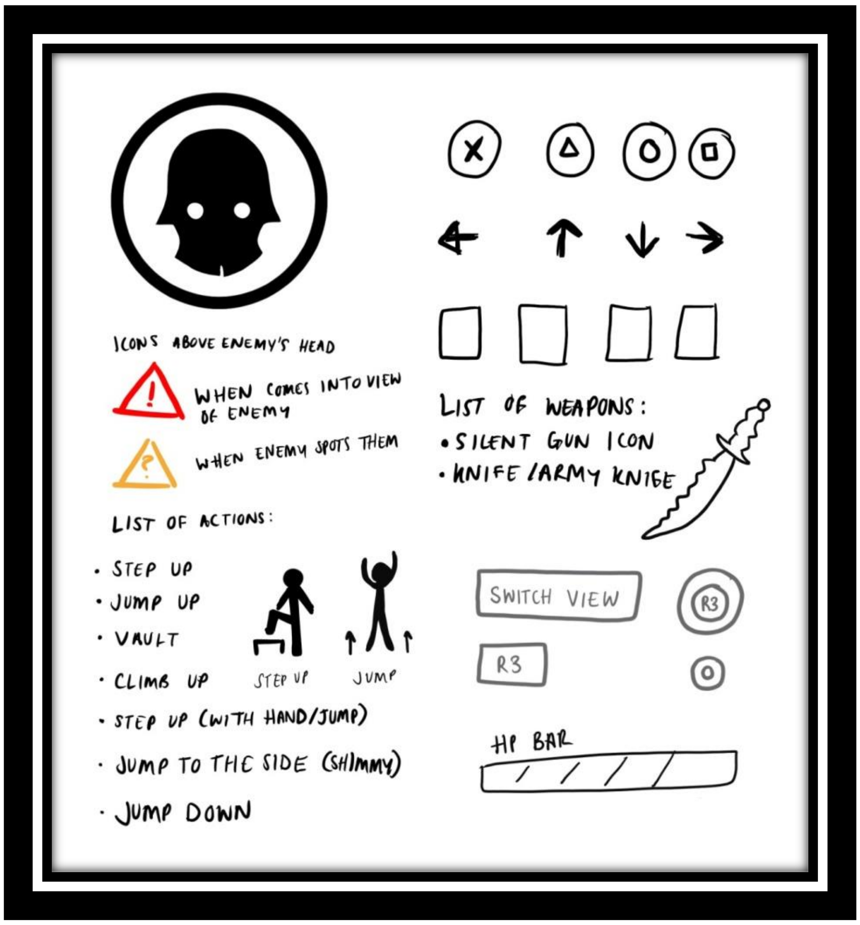
The game aims to have a UI that offers the player all the necessary information during the game. In one corner it will have four quick-access slots for items such as weapons or gadgets that the player has equipped in those spaces. Just below a health bar indicating the player's life and could even incorporate a stamina bar that decreases when the player sprints or climbs.

In the right corner there is a radar that could help locate nearby enemies while the two bars above it would be indicators showing the level of light exposure and the level of noise the player is generating.

In the centre, messages would appear indicating to the player when it is possible to perform an action such as opening a door, reloading ammo, performing a parkour action, climbing, and any other interaction in the level.



In-game UI sketch



Sketches of icons that could appear on the screen to perform actions and other indicators.

Level Design

The level design is intended to emulate the Icelandic coastline as it is a requirement of the narrative that events take place in this part of the world. The level begins with a cinematic that introduces the mission to the player who can listen a conversation between the protagonist and an allied colonel.

The game starts at the bottom of a cliff with no enemies. This is a fact that is not accidental since the intention of being alone on the cliff is to give players the opportunity to experience the controls. Some objects have been placed so that the player can intuitively discover the Parkour System.

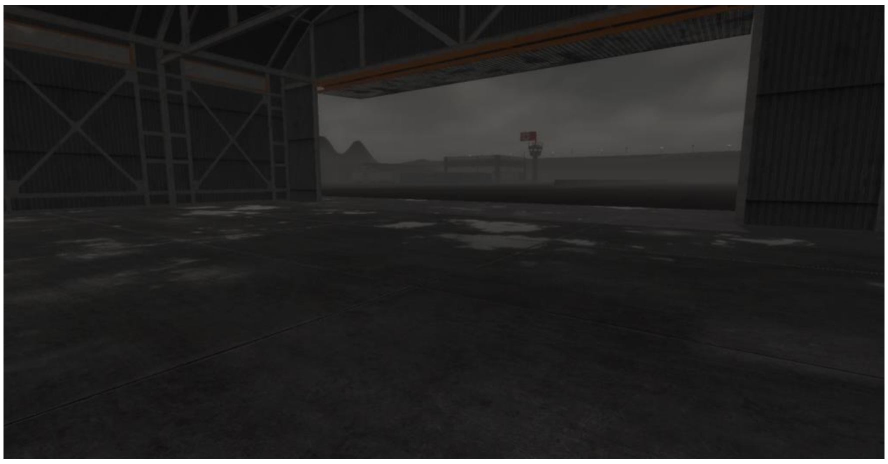
Another reason why the starting point is at the bottom of a cliff is to also introduce the climbing mechanic as soon as possible so that the player realizes that there are a variety of mechanics and can become familiar with them.

Once they climb the cliff, there is a wooded area with grass and trees that will serve to hide from some enemies that patrol that area. In this way, the stealth mechanics that characterizes the game is introduced.

From this point on, the player has the option of going to the access door guarded by a guard or going down another cliff where they will meet the walls of the enemy base.

The first option is designed so that the player understands that they must cause the guard to leave his post with the noise mechanic.

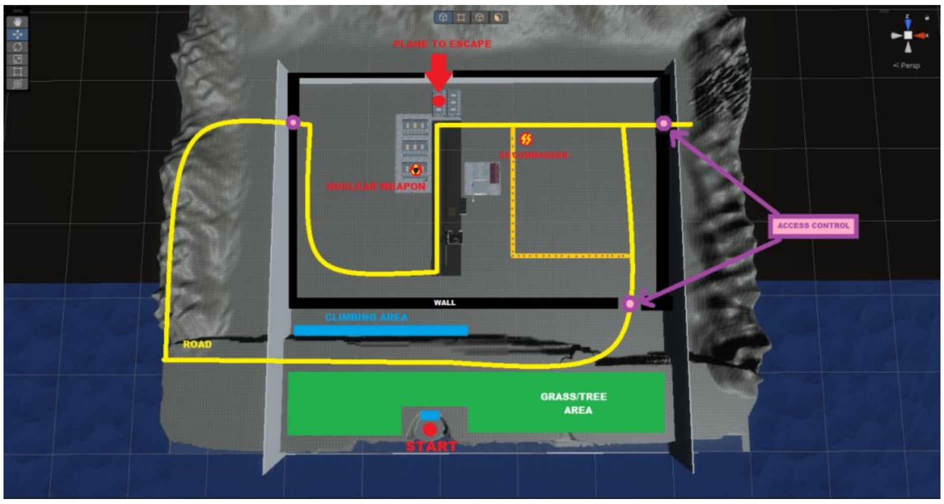
On the other hand, the other option invites exploration and finding a conduit that allows the player to cross the walls of the underground enemy base.



Once inside the base, there will be many more enemies and other dangers such as light sources and surveillance cameras.

At this point the player has to cautiously explore the level to find the objectives although cinematic will be triggered that give the player clues where to go. There will also be the option to contact by radio, which will consist of pressing a button so that the colonel reminds us of the next objective.

Once all the objectives have been met, the player must go to a hangar and get on a plane to escape, closing the game's narrative with a tense action scene full of enemies shooting.



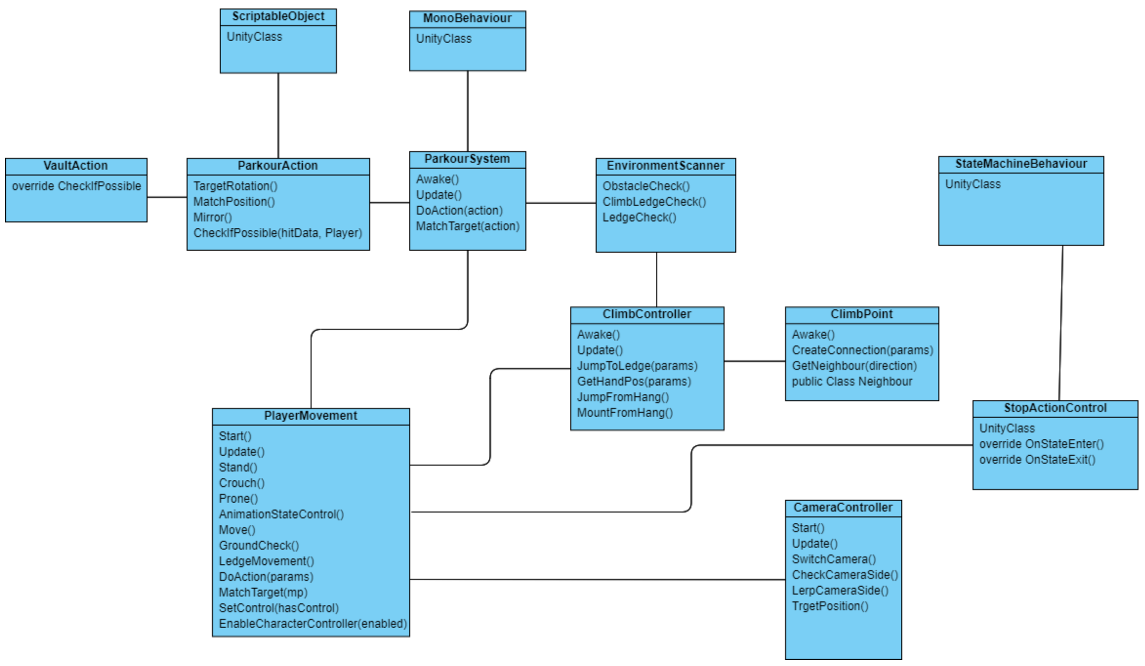
Technical Design

The present diagram demonstrates the current state of technical design employed in the project concerning implementing the essential mechanics required for the complete movement of the playable character. The diagram shows the flow of data between various scripts which enable the character to navigate through the level by utilizing the Parkour System and Climbing System.

The diagram outlines the course of data flow leading to the PlayerMovement script, responsible for controlling the character's motion, administering animations, and managing the diverse stances, such as from standing to crouching and other states.

The Parkour System is comprised of three scripts, implemented as classes inheriting from distinct parent classes. The Parkour controller inherits from the Monobehaviour class, whereas the ParkourAction class inherits from the ScriptableObject class. Additionally, the VaultAction class, which only overrides the CheckIfPossible() function, inherits from the ParkourAction class.

Conversely, the Climbing System comprises two scripts, namely the ClimbController and the ClimbPoint. Both systems interact with the EnvironmentScanner script, which utilizes raycasts to detect interactable objects present in the scene.

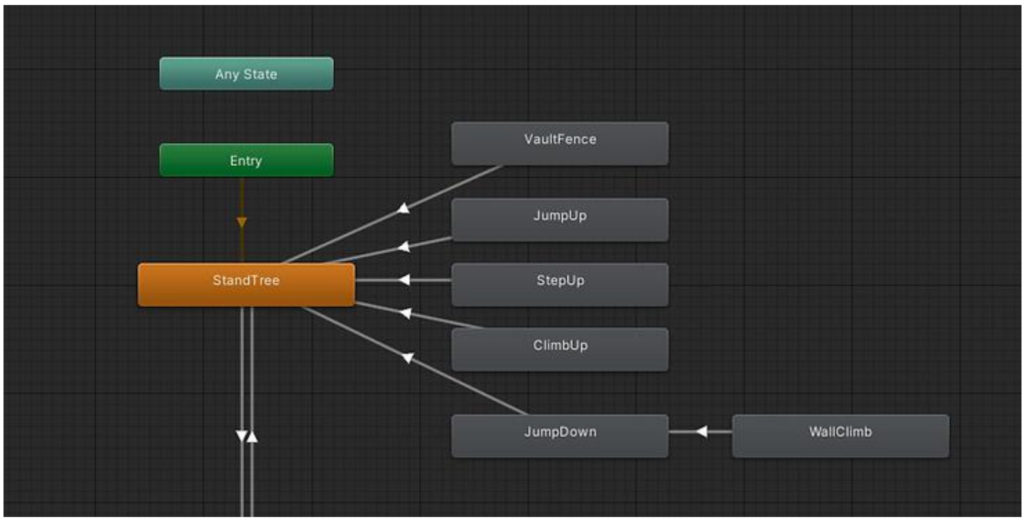


5. Implementation

The most prominent implementation of this project so far is the Parkour System and the Climbing System.

Parkour System

The Parkour System is a series of animated actions that the player can perform by pressing a button in front of different types of objects. Some of these actions are “StepUp”, “JumpUp”, “ClimbUp”, “VaultFence” and “WallClimb”.

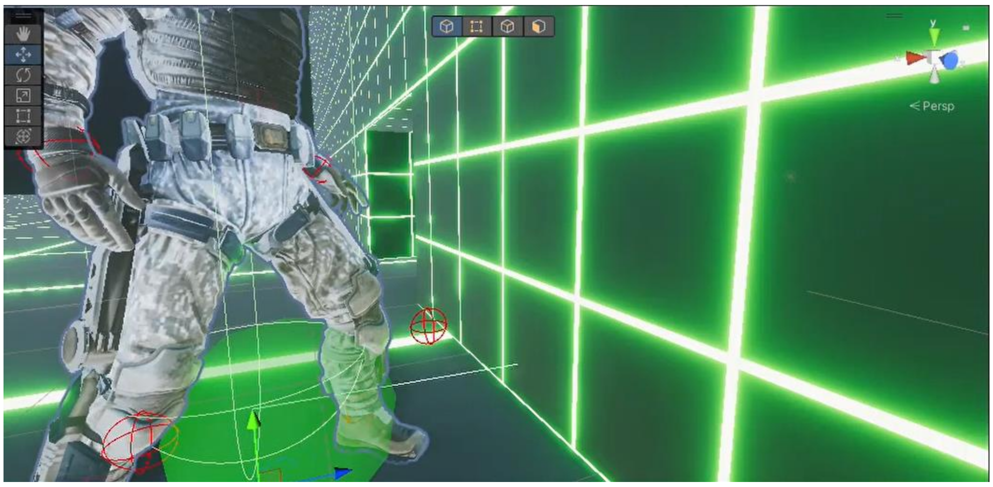


Each action has a different animation and these are triggered depending on the minimum and maximum height of each object. In addition to distinguishing objects by height, it also identifies objects by Layers and Tags since there are objects that have a similar height range but require different animation.

For this, the player character needs to communicate with the environment and receive and manage the different data it receives from the environment. This is achieved with the implementation of a scanner that allows the player to scan the environment thanks to the raycast, which informs through a boolean if it has hit an object or not which must have a LayerMask called "Environment".

For this, two scripts are initially needed, one that scans the scenario and sends the received data, and another that receives this data to determine what to do with it.

The EnvironmentScanner Script contains the public function ObstacleCheck() which is called from the ParkourController Script. This function has an instance of a data holder of type struct that sends data to determine if the object has been raycast. The ObstacleCheck() function also checks the height of the struck object. It does this by casting a second raycast up from the point where the object was hit when the first raycast returns true. This second raycast has a greater length and determines what maximum height it scans, in this case, the length of the second raycast is 5 meters.



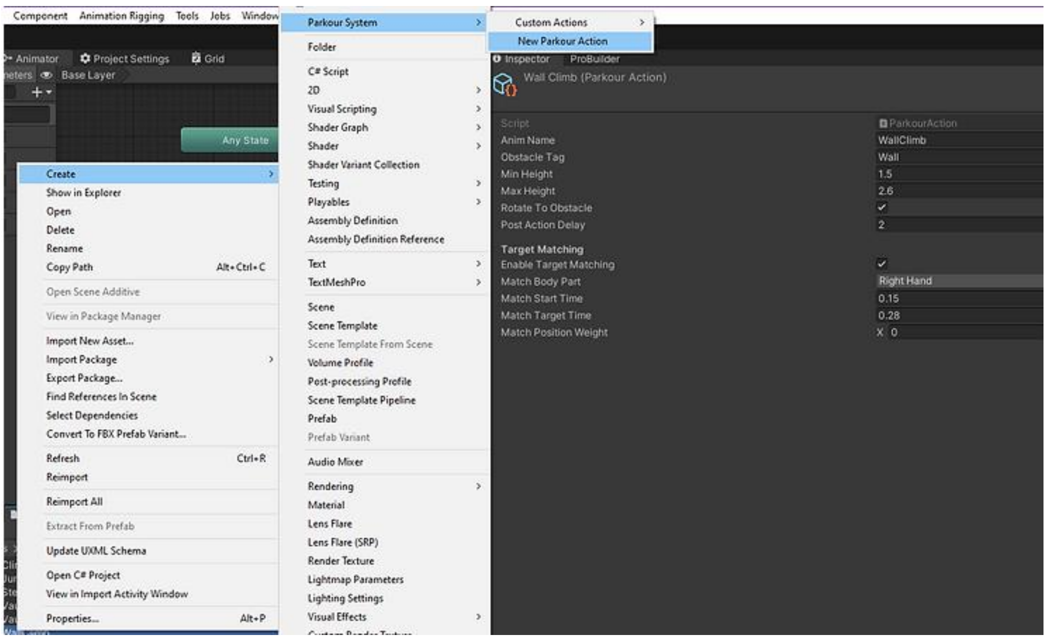
The ParkourController Script checks in its Update function if this boolean is true or false. If true then it activates the parkour action, telling the Animator Component which animation to play and briefly disabling the Move() function of the PlayerController script which will prevent the player from moving while the action is taking place. Notably, the Parkour Controller script also disables the Character Controller Component to prevent the collider or gravity from affecting the animation.

Scriptable Objects

The different animations based on the height of an object are triggered by creating a scriptable object for each action the player character needs to do. A scriptable Object is a data container that allows the storage of the necessary information for each animation. In this way, objects that contain a certain animation with certain parameters can be created without the need to be coding each parameter for each animation in a Monobehaviour script.

To start implementing different actions in the parkour system, it is necessary to create a script called ParkourAction inherited from the ScritableObject class. This script contains the data, variables, and parameters necessary to play any animation related to the Parkour System. This script specifies data such as the name of the animation, the minimum, and maximum height at which each animation must be triggered, the label of the obstacle, booleans so that the agent can rotate or not depending on the animation, or even the option to add an animation post-delay action to regain control of the player character in case multiple animations are combined and it is necessary to wait for the animations to complete before the player can move the character again.

Once implemented, it is possible to create ParkourAction instances from the Unity editor.



Wall Climb (Parkour Action)

Script	ParkourAction
Anim Name	WallClimb
Obstacle Tag	Wall
Min Height	1.5
Max Height	2.6
Rotate To Obstacle	✓
Post Action Delay	2
Target Matching	
Enable Target Matching	✓
Match Body Part	Right Hand
Match Start Time	0.15
Match Target Time	0.28
Match Position Weight	X 0

After creating different actions, the ParkourController script must handle them. To do this, it is necessary to declare an action list in the el script. The ParkourController will have to iterate through each action every time the ObstacleCheck() function returns true and the button is pressed.

To determine that an action is possible, the ParkourAction script contains a function called CheckIfPossible() that takes as parameters the data obtained by the raycast and the player transform. This function calculates the height of the obstacle by subtracting the distance between the Y position of the height hit point by the Y position of the player. If the calculated height is between the minimum and maximum values assigned to the asset, then the action is possible. But in case obstacles of the same height require different actions, it is necessary to apply a tag to the obstacle and assign said tag to the asset so that when the ParkourAction script compares the string it knows what the object is.

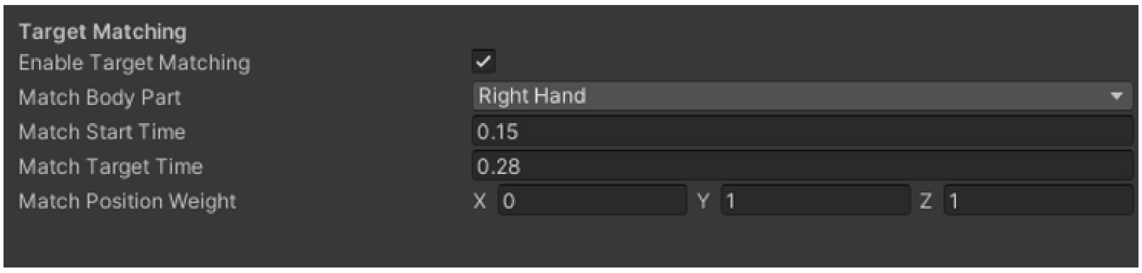
So a layer mask is used to determine which objects are interactive and tags to determine the obstacle type, for instance, a step or a fence with the same range of height.

Target Matching technique

At this point, the player character can position himself on top of different obstacles with different heights playing different animations but there is still a problem. And it is that the animations are generic and always reproduce the same movement without adapting well to the measurements of the obstacle, so there are clipping problems, that the hand does not position itself on the edge of the obstacle or the foot stops before touching soil. To work around this problem, Unity provides a method known as Target Matching.

With this technique, it is possible to achieve that the animations match obstacles of different dimensions without the need to implement different animations for each obstacle.

To implement Target Matching, it is necessary to expand the ParkourAction script by adding float variables that determine at which frame of the animation a certain body part is required to match the desired position. In some cases target matching will not be required, for this reason, it is also necessary to apply a boolean that determines whether or not an asset should apply target matching. You also need a variable of the AvatarTarget type to be able to select which part of the body you want to match with the obstacle in a certain frame. The AvatarTarget is an enum that communicates with parts of the rig like root, body, left foot, right foot, left hand, and right hand. It also has a variable of type Vector3 that allows you to match and correct some animation position definitions in order to avoid the clipping effect with obstacles. All these variables are accessible through the asset inspector created by the ParkourAction script.


Target Matching

Enable Target Matching	✓
Match Body Part	Right Hand
Match Start Time	0.15
Match Target Time	0.28
Match Position Weight	X 0 Y 1 Z 1

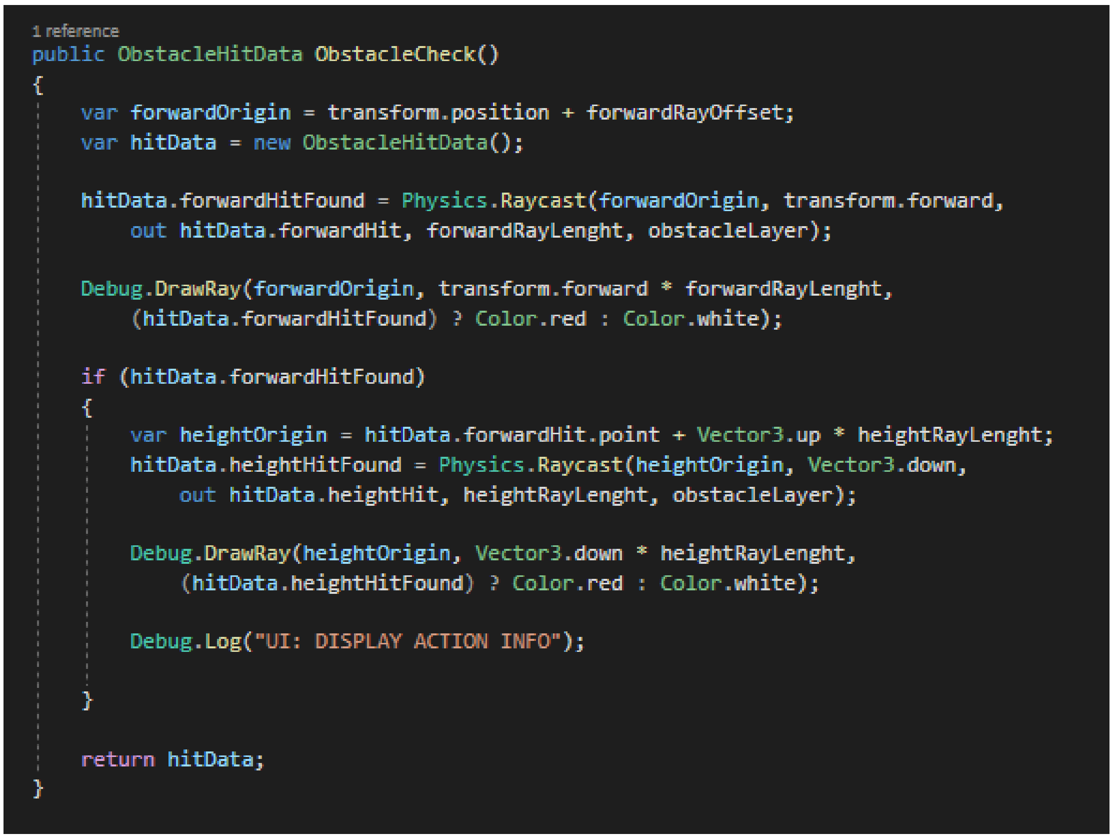
To know which frame should be set as Match Start Time and Match Target Time, you have to select the desired animation and inspect it in the inspector timeline. For example, the Match Start Time would be the frame where the animation begins to jump, 15% of the animation, while the Match Target Time would be the frame where you want to place the desired body part, a value of approximately 33% animation. These percentage values must be normalized, so the final value to be assigned to the action asset would be 0.15 and 0.28.

Code Snippets

Environment Scanner script

The EnvironmentScanner script contains three methods to detect obstacles, ledges, and climbable ledges in the environment around the player character.

The ObstacleCheck() method checks for obstacles in front of the game object, and if an obstacle is detected, it performs a secondary check to determine if the obstacle can be climbed. This method returns a struct called ObstacleHitData containing information about the detected obstacle and its climbability.



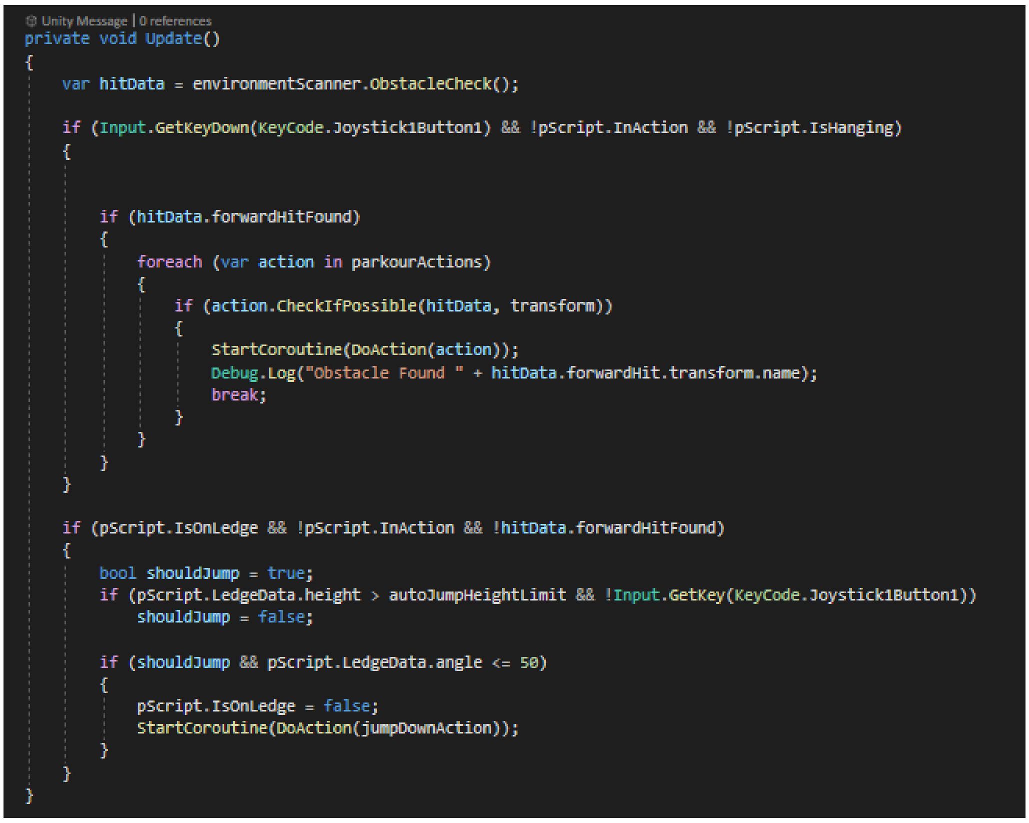
```
1 reference
public ObstacleHitData ObstacleCheck()
{
    var forwardOrigin = transform.position + forwardRayOffset;
    var hitData = new ObstacleHitData();

    hitData.forwardHitFound = Physics.Raycast(forwardOrigin, transform.forward,
        out hitData.forwardHit, forwardRayLenght, obstacleLayer);

    Debug.DrawRay(forwardOrigin, transform.forward * forwardRayLenght,
        (hitData.forwardHitFound) ? Color.red : Color.white);

    if (hitData.forwardHitFound)
    {
        var heightOrigin = hitData.forwardHit.point + Vector3.up * heightRayLenght;
        hitData.heightHitFound = Physics.Raycast(heightOrigin, Vector3.down,
            out hitData.heightHit, heightRayLenght, obstacleLayer);

        Debug.DrawRay(heightOrigin, Vector3.down * heightRayLenght,
            (hitData.heightHitFound) ? Color.red : Color.white);

        Debug.Log("UI: DISPLAY ACTION INFO");

    }

    return hitData;
}
```

Parkour System script

The ParkourSystem script contains variables and methods that are used to enable the player to perform parkour actions in the game. The script relies on the "EnvironmentScanner" script to detect obstacles in the player's path and the "PlayerMovement" script to control the player's movement and actions.

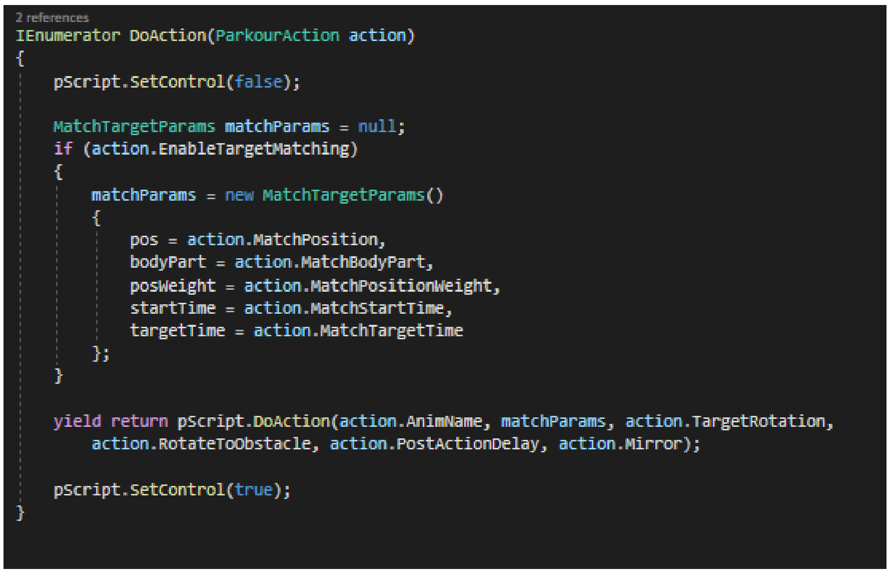
The "Update" method is called every frame and contains code to detect player input and obstacles in the player's path. If the player presses a specific button and is not currently performing another action or hanging from a ledge, the script checks if there is an obstacle in front of the player and if any of the parkour actions in the list of "parkourActions" are possible based on the player's position and the obstacle's characteristics. If a parkour action is possible, the script starts a coroutine to execute the action.

If the player is currently on a ledge and there is no obstacle in front of them, the script checks if the player should automatically jump down based on the height of the ledge and whether the player is pressing the jump button. If the conditions are met, the script starts a coroutine to execute the "jumpDownAction".



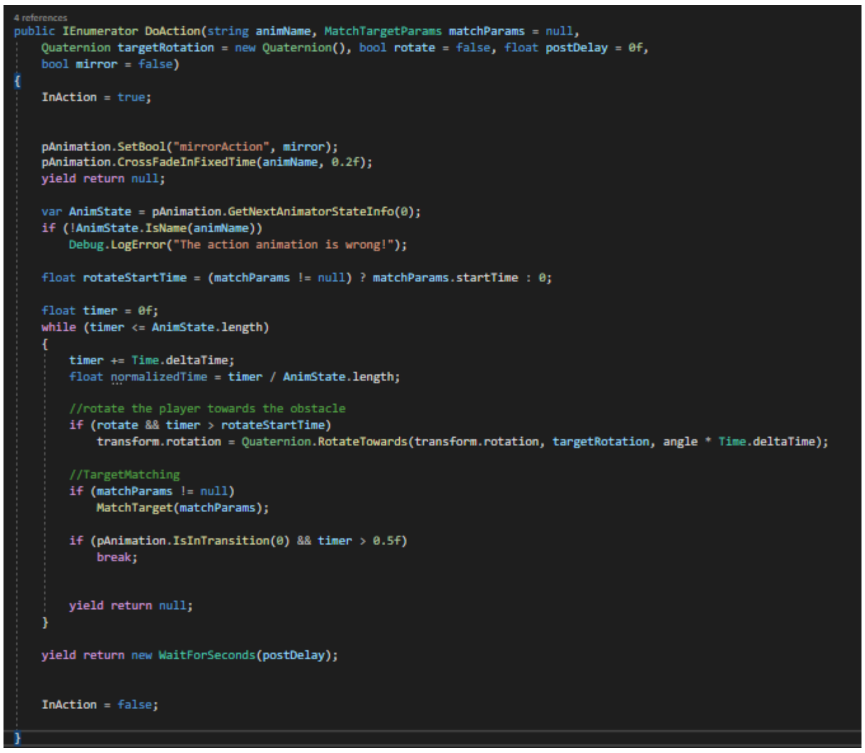
```
Unity Message | 0 references
private void Update()
{
    var hitData = environmentScanner.ObstacleCheck();

    if (Input.GetKeyDown(KeyCode.Joystick1Button1) && !pScript.InAction && !pScript.IsHanging)
    {


        if (hitData.forwardHitFound)
        {
            foreach (var action in parkourActions)
            {
                if (action.CheckIfPossible(hitData, transform))
                {
                    StartCoroutine(DoAction(action));
                    Debug.Log("Obstacle Found " + hitData.forwardHit.transform.name);
                    break;
                }
            }
        }
    }

    if (pScript.IsOnLedge && !pScript.InAction && !hitData.forwardHitFound)
    {
        bool shouldJump = true;
        if (pScript.LedgeData.height > autoJumpHeightLimit && !Input.GetKey(KeyCode.Joystick1Button1))
            shouldJump = false;

        if (shouldJump && pScript.LedgeData.angle <= 50)
        {
            pScript.IsOnLedge = false;
            StartCoroutine(DoAction(jumpDownAction));
        }
    }
}
```

The "DoAction" co-routine is used to execute a parkour action. It temporarily disables the player's control, plays the appropriate animation, and then re-enables the player's control. It also supports target matching, which adjusts the position and rotation of the player during the animation to match a specific position or object in the game world.



```
2 references
IEnumerator DoAction(ParkourAction action)
{
    pScript.SetControl(false);

    MatchTargetParams matchParams = null;
    if (action.EnableTargetMatching)
    {
        matchParams = new MatchTargetParams()
        {
            pos = action.MatchPosition,
            bodyPart = action.MatchBodyPart,
            posWeight = action.MatchPositionWeight,
            startTime = action.MatchStartTime,
            targetTime = action.MatchTargetTime
        };
    }

    yield return pScript.DoAction(action.AnimName, matchParams, action.TargetRotation,
        action.RotateToObstacle, action.PostActionDelay, action.Mirror);

    pScript.SetControl(true);
}
```

This co-routine function is used to play an animation and perform specific actions during the animation playback. It takes in several parameters such as the name of the animation, whether to rotate the player towards the obstacle, delay after the action is completed, etc. During the execution of the co-routine, it sets a bool flag InAction to true, plays the animation, rotates the player if needed, and performs target matching if specified. Once the animation is finished, it waits for the specified post-delay and sets the InAction flag to false.

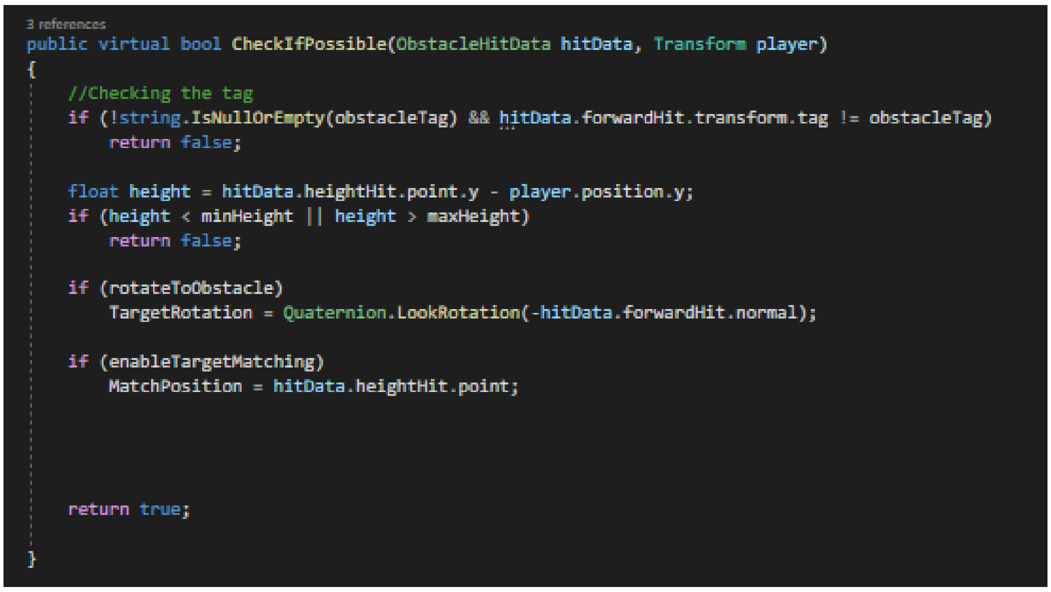


```
4 references
public IEnumerator DoAction(string animName, MatchTargetParams matchParams = null,
    Quaternion targetRotation = new Quaternion(), bool rotate = false, float postDelay = 0f,
    bool mirror = false)
{
    InAction = true;


    pAnimation.SetBool("mirrorAction", mirror);
    pAnimation.CrossFadeInFixedTime(animName, 0.2f);
    yield return null;

    var AnimState = pAnimation.GetNextAnimatorStateInfo(0);
    if (!AnimState.IsName(animName))
        Debug.LogError("The action animation is wrong!");

    float rotateStartTime = (matchParams != null) ? matchParams.startTime : 0;

    float timer = 0f;
    while (timer <= AnimState.length)
    {
        timer += Time.deltaTime;
        float normalizedTime = timer / AnimState.length;

        //rotate the player towards the obstacle
        if (rotate && timer > rotateStartTime)
            transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotation, angle * Time.deltaTime);

        //TargetMatching
        if (matchParams != null)
            MatchTarget(matchParams);

        if (pAnimation.IsInTransition(0) && timer > 0.5f)
            break;


        yield return null;
    }

    yield return new WaitForSeconds(postDelay);


    InAction = false;
}
```

Parkour Action script

This code defines a scriptable object called "ParkourAction" that is used in the parkour system. It contains various properties related to the action such as the animation name, obstacle tag, minimum and maximum height required to perform the action, whether the player needs to rotate to face the obstacle, post-action delay time, and target matching related properties.

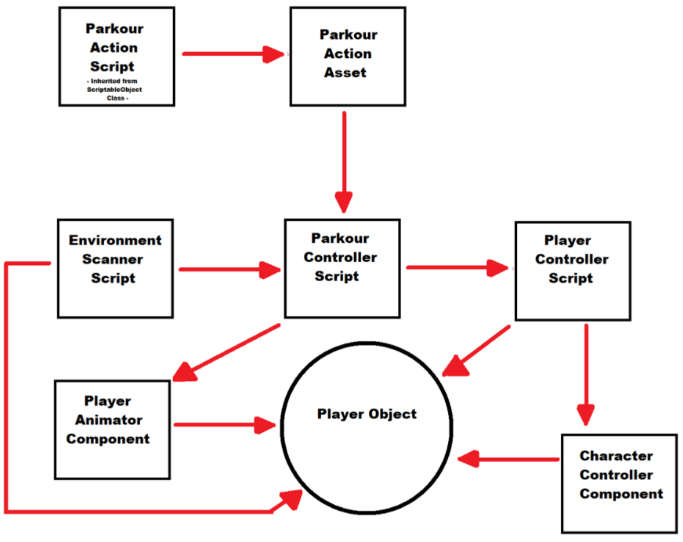
The method "CheckIfPossible" checks if the parkour action is possible given the obstacle hit data and player's position. It returns a boolean value indicating if the action is possible.



```
3 references
public virtual bool CheckIfPossible(ObstacleHitData hitData, Transform player)
{
    //Checking the tag
    if (!string.IsNullOrEmpty(obstacleTag) && hitData.forwardHit.transform.tag != obstacleTag)
        return false;

    float height = hitData.heightHit.point.y - player.position.y;
    if (height < minHeight || height > maxHeight)
        return false;

    if (rotateToObstacle)
        TargetRotation = Quaternion.LookRotation(-hitData.forwardHit.normal);

    if (enableTargetMatching)
        MatchPosition = hitData.heightHit.point;




    return true;

}
```

Other properties such as TargetRotation, MatchPosition, and Mirror are used to store the values of the target rotation, match position, and mirror state of the action, respectively.

Parkour System Architecture

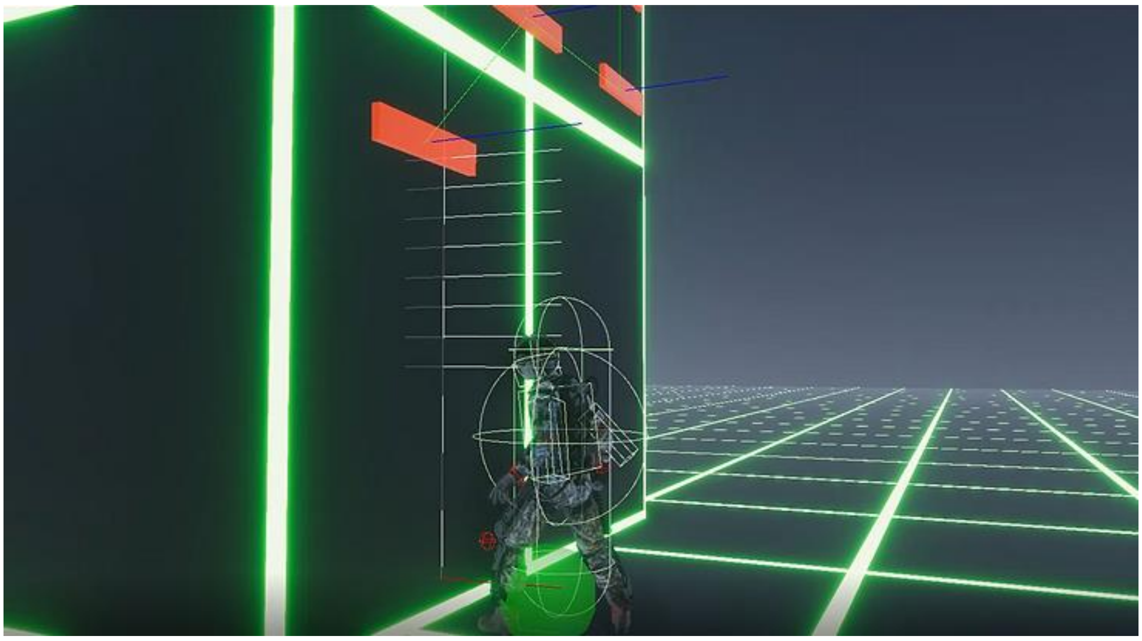


Parkour Action Script
- Inherited from ScriptableObject Class -
Parkour Action Asset
Environment Scanner Script
Parkour Controller Script
Player Controller Script
Player Animator Component
Player Object
Character Controller Component

Climbing System

To implement the climbing system, the first thing you need to do is create climbing objects and put a specific tag on them so that the raycast can differentiate between climbing and environmental objects related to the parkour system.

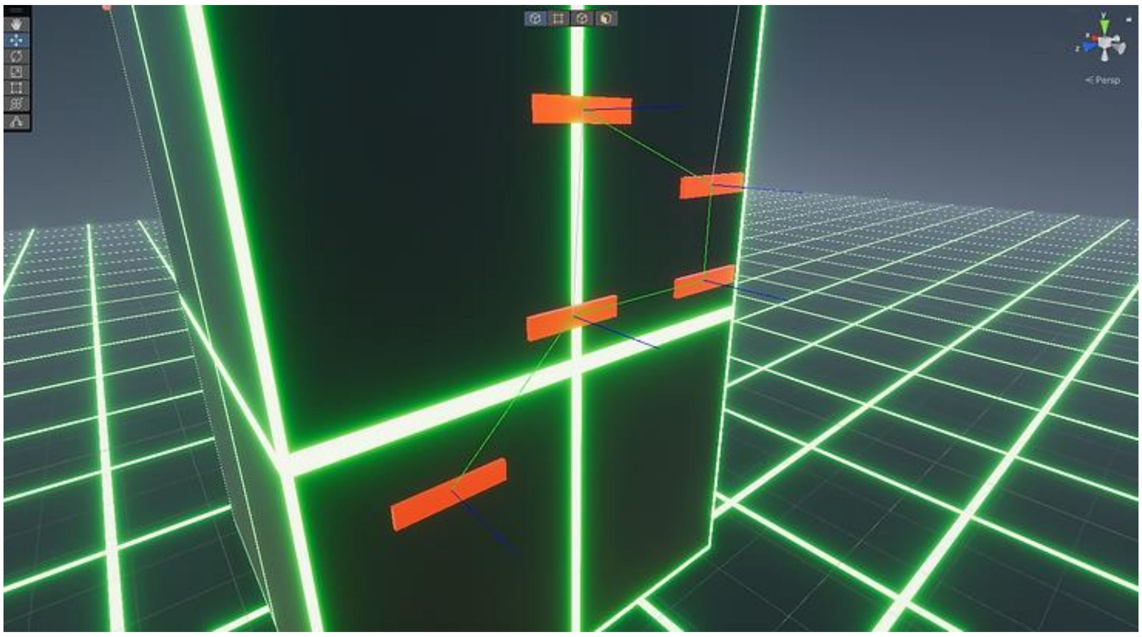
To detect these new objects in the scene, it is necessary for the scanner to detect them with another series of raycasts that have been implemented in the upper part of the character so that it can detect ledges at different heights.



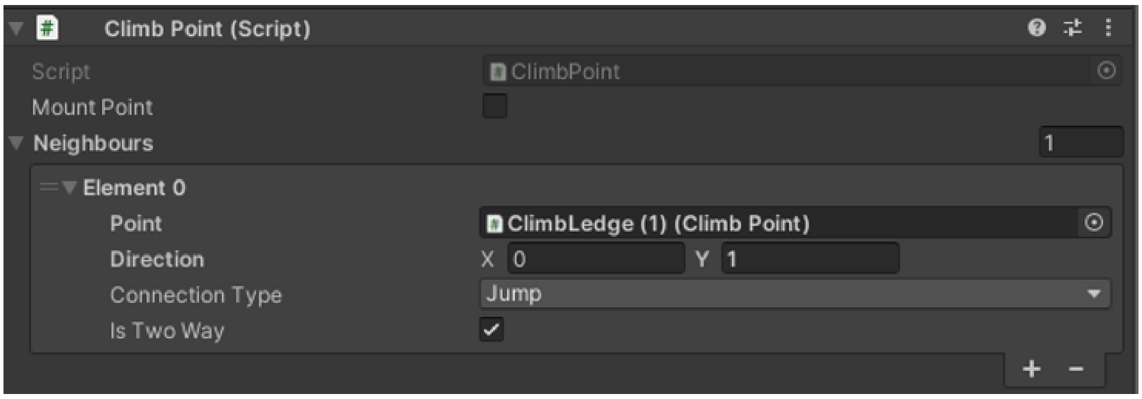
Once it detects edges, the player character jumps and hangs until new input is received. To do this, target matching technique is used so the character's animation feels realistic and his hands are well placed to the edge. While the character is hanging and preventing another movement code from being executed, a boolean (isHanging) is applied which determines whether or not the character is climbing edges.

To make the character jump from one edge to another, a climbing network has been implemented that allows communicating edges, and the character can only move within this network while in the hanging state.

The ledges are joined by two types of lines, green and white. The green ones indicate that this path can be transitioned in both directions while with the white ones, the character can only move in one direction. In other words, the player character is able to move from a high edge to a lower edge but not the other way around if the previous edge is too high.



This is achieved by adding a script to each ledge called a climbPoint that indicates which direction the character can move with a Vector2. This script also has a list of neighboring edges where the character can jump to or move from the current climbPoint. Therefore two types of climbPoints are needed, jumpable ones and moveable ones in case an edge has two or more climbPoints. This script also determines if the climbPoint where the character is is a point that the character can mount or not. One of the limitations of this system is that a climbPoint can only have a maximum of 4 connections when moving between climbPoints. With a Vector2 the number of directions is limited to the right (X = 1), left (X = -1), up (Y = 1), and down (Y = -1).



Climb Point (Script)

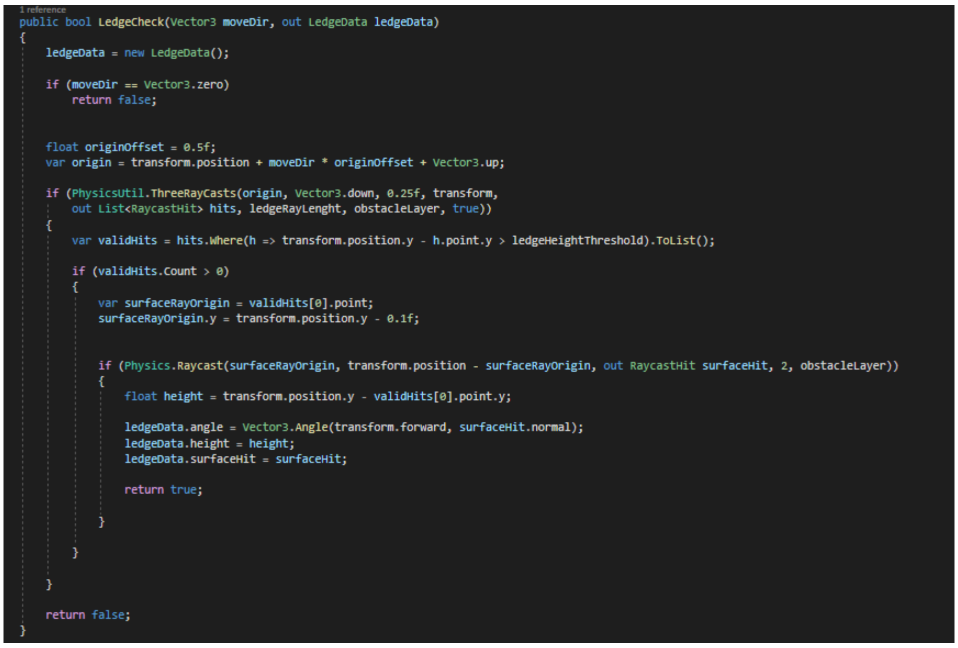
Script	ClimbPoint
Mount Point	☐
Neighbours	1
Element 0	
Point	ClimbLedge (1) (Climb Point)
Direction	X 0 Y 1
Connection Type	Jump
Is Two Way	✓

Once the climbing network has been set, and the character can jump to the first climbPoint, another input direction has been coded in the Climbing Controller script for when the character is in the isHanging state. This input detects in which direction the player wants to move and if there is a climbPoint in that direction it will jump or move, depending on its type, towards the neighboring climbPoint and convert the neighboring climbPoint to the current climbPoint to determine which is the new neighboring climbPoint.

Code Snippets

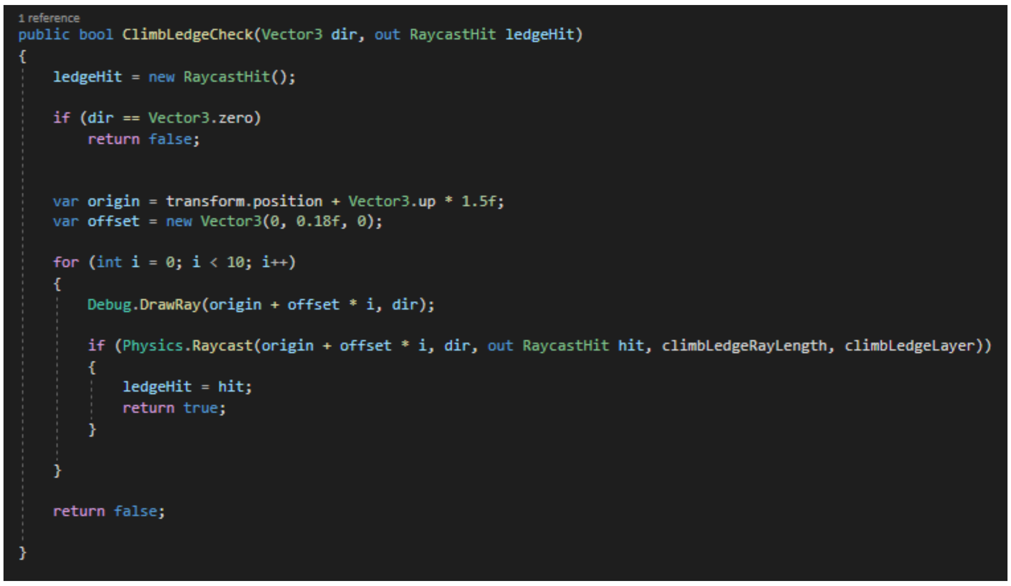
Environment Scanner script

The LedgeCheck() method checks if there is a ledge within a certain range in the direction that the game object is moving. This method returns a struct called LedgeData containing information about the height and angle of the detected ledge.



```
1 reference
public bool LedgeCheck(Vector3 moveDir, out LedgeData ledgeData)
{
    ledgeData = new LedgeData();

    if (moveDir == Vector3.zero)
        return false;


    float originOffset = 0.5f;
    var origin = transform.position + moveDir * originOffset + Vector3.up;

    if (PhysicsUtil.ThreeRayCasts(origin, Vector3.down, 0.25f, transform,
        out List<RaycastHit> hits, ledgeRayLenght, obstacleLayer, true))
    {
        var validHits = hits.Where(h => transform.position.y - h.point.y > ledgeHeightThreshold).ToList();

        if (validHits.Count > 0)
        {
            var surfaceRayOrigin = validHits[0].point;
            surfaceRayOrigin.y = transform.position.y - 0.1f;


            if (Physics.Raycast(surfaceRayOrigin, transform.position - surfaceRayOrigin, out RaycastHit surfaceHit, 2, obstacleLayer))
            {
                float height = transform.position.y - validHits[0].point.y;

                ledgeData.angle = Vector3.Angle(transform.forward, surfaceHit.normal);
                ledgeData.height = height;
                ledgeData.surfaceHit = surfaceHit;

                return true;

            }

        }

    }

    return false;
}
```

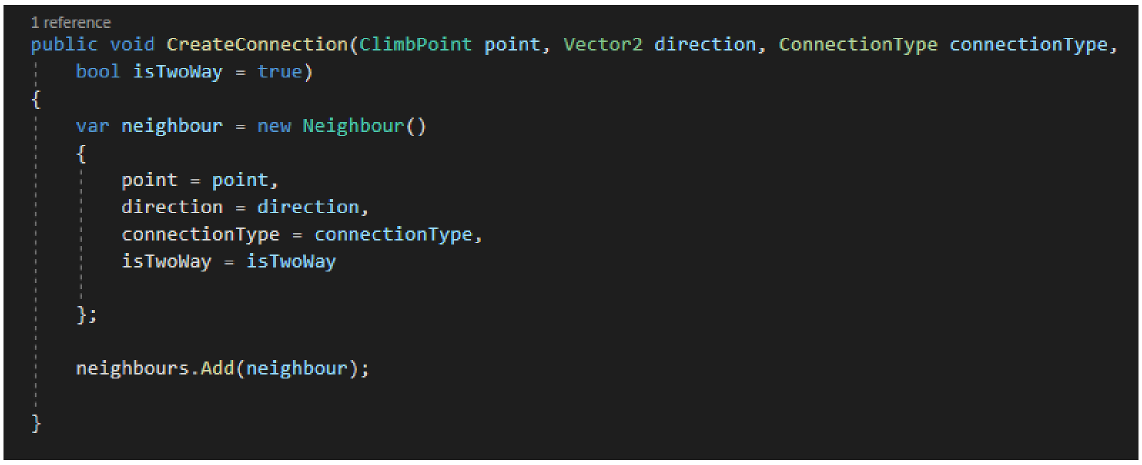
The ClimbLedgeCheck() method checks if a climbable ledge is within range in the specified direction. This method returns a boolean indicating whether a climbable ledge is detected, and it also outputs a RaycastHit struct containing information about the detected ledge.



```
1 reference
public bool ClimbLedgeCheck(Vector3 dir, out RaycastHit ledgeHit)
{
    ledgeHit = new RaycastHit();

    if (dir == Vector3.zero)
        return false;


    var origin = transform.position + Vector3.up * 1.5f;
    var offset = new Vector3(0, 0.18f, 0);

    for (int i = 0; i < 10; i++)
    {
        Debug.DrawRay(origin + offset * i, dir);

        if (Physics.Raycast(origin + offset * i, dir, out RaycastHit hit, climbLedgeRayLength, climbLedgeLayer))
        {
            ledgeHit = hit;
            return true;
        }

    }

    return false;

}
```

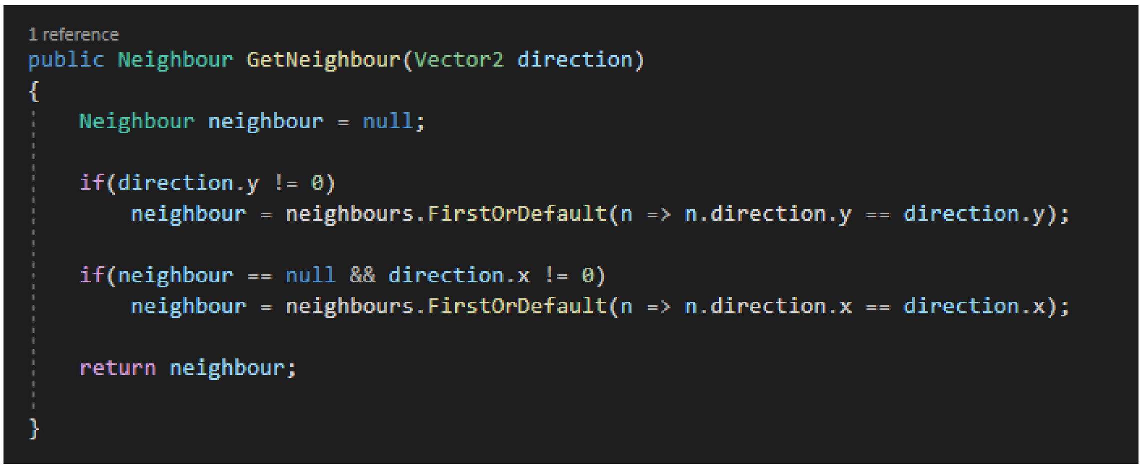
Climb Point script

The CreateConnection method of the ClimbPoint class is used to add a new Neighbour to the neighbours list. A Neighbour object represents a connection to another ClimbPoint, along with a direction vector, a connection type (either Jump or Move), and whether the connection is two-way.



```
1 reference
public void CreateConnection(ClimbPoint point, Vector2 direction, ConnectionType connectionType,
    bool isTwoWay = true)
{
    var neighbour = new Neighbour()
    {
        point = point,
        direction = direction,
        connectionType = connectionType,
        isTwoWay = isTwoWay

    };

    neighbours.Add(neighbour);

}
```

The GetNeighbour method is used to retrieve a Neighbour object from the neighbours list based on a given direction vector.

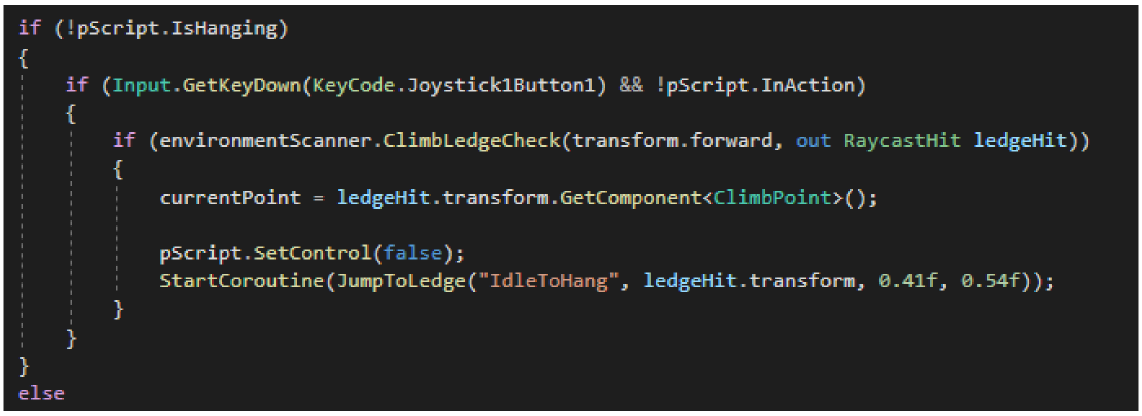


```
1 reference
public Neighbour GetNeighbour(Vector2 direction)
{
    Neighbour neighbour = null;

    if(direction.y != 0)
        neighbour = neighbours.FirstOrDefault(n => n.direction.y == direction.y);

    if(neighbour == null && direction.x != 0)
        neighbour = neighbours.FirstOrDefault(n => n.direction.x == direction.x);

    return neighbour;

}
```

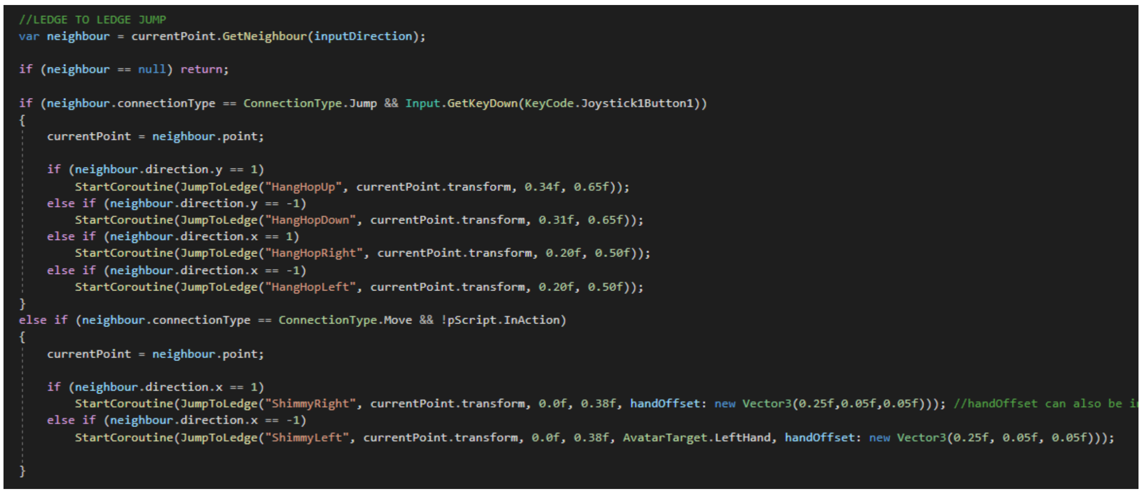
Climb Controller script

This script is used to handle player climbing mechanics. It uses an "EnvironmentScanner" component to detect climbable objects, and a "PlayerMovement" component to control the player's movement.

When the player is not hanging from a ledge, the script listens for input to trigger a jump to a nearby climbable ledge, and when the player is hanging, it listens for input to perform various actions such as jumping to another ledge or shimmying along the current one.



```
if (!pScript.IsHanging)
{
    if (Input.GetKeyDown(KeyCode.Joystick1Button1) && !pScript.InAction)
    {
        if (environmentScanner.ClimbLedgeCheck(transform.forward, out RaycastHit ledgeHit))
        {
            currentPoint = ledgeHit.transform.GetComponent<ClimbPoint>();

            pScript.SetControl(false);
            StartCoroutine(JumpToLedge("IdleToHang", ledgeHit.transform, 0.41f, 0.54f));
        }
    }
}
else
```



```
//LEDGE TO LEDGE JUMP
var neighbour = currentPoint.GetNeighbour(inputDirection);

if (neighbour == null) return;

if (neighbour.connectionType == ConnectionType.Jump && Input.GetKeyDown(KeyCode.Joystick1Button1))
{
    currentPoint = neighbour.point;

    if (neighbour.direction.y == 1)
        StartCoroutine(JumpToLedge("HangHopUp", currentPoint.transform, 0.34f, 0.65f));
    else if (neighbour.direction.y == -1)
        StartCoroutine(JumpToLedge("HangHopDown", currentPoint.transform, 0.31f, 0.65f));
    else if (neighbour.direction.x == 1)
        StartCoroutine(JumpToLedge("HangHopRight", currentPoint.transform, 0.20f, 0.50f));
    else if (neighbour.direction.x == -1)
        StartCoroutine(JumpToLedge("HangHopLeft", currentPoint.transform, 0.20f, 0.50f));
}
else if (neighbour.connectionType == ConnectionType.Move && !pScript.InAction)
{
    currentPoint = neighbour.point;

    if (neighbour.direction.x == 1)
        StartCoroutine(JumpToLedge("ShimmyRight", currentPoint.transform, 0.0f, 0.38f, handOffset: new Vector3(0.25f,0.05f,0.05f))); //handOffset can also be i
    else if (neighbour.direction.x == -1)
        StartCoroutine(JumpToLedge("ShimmyLeft", currentPoint.transform, 0.0f, 0.38f, AvatarTarget.LeftHand, handOffset: new Vector3(0.25f, 0.05f, 0.05f)));

}
```

The script uses co-routines to handle animations during these actions, and includes functions to calculate and set the position of the player's hands during animations.

6. Testing and Evaluation

Testing and evaluation are critical aspects of any software development project. They ensure that the software meets the requirements and functions as intended. This section provides an overview of the testing and evaluation process for the project, including test case design, debugging, bugs found, player feedback, and evaluation.

Periodic testing has been performed on the project following the implementation of new features or scenarios to guarantee their expected functionality. The project has experienced some performance issues at certain times due to an increase in content. To address this, optimization efforts were made incrementally, although these were not exhaustive due to time constrains.

Below are some test cases that have been carried out with different mid-range and high-end equipment. These cases cover functionalities such as the introductory video of the game, the main menu, the game cinematics and its transitions between scenes and finally the fundamental implementations of this project, the Parkour System and the Climbing System.

Test Cases

Test Case #1

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check video functionality	Check video frame rate and sound	Video plays smoothly and sound plays properly	Video plays smoothly and sound plays properly	PASS
	Check if video can be skipped through input	Game transitions to the Main Menu Screen	Game transitions to the Main Menu Screen	PASS
	Check transition to Main Menu	Game transitions to the Main Menu Screen	Game transitions to the Main Menu Screen	PASS

Test Case #2

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check Main Menu functionality	Check navigation through Gamepad input	Navigation responds to gamepad input and highlight the buttons	Navigation responds to gamepad input and highlight the buttons	PASS
	Check New Game button	App transitions to the loading screen and plays cinematic.	App transitions to the loading screen and plays cinematic.	PASS
	Check Training button	App transitions to the loading screen and starts training level	App transitions to the loading screen and starts training level	PASS
	Check Exit button	app closes correctly	app closes correctly	PASS

Test Case #3

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check cinematic functionality	Check cinematic frame rate and sound	Cinematic plays smoothly and sound plays properly	Cinematic plays smoothly and sound plays properly	PASS
	Check if cinematic can be skipped through input	Cinematic is skipped and user has control over the charecter	Cinematic is skipped and user has control over the character	PASS
	Check any graphical bug or issue	Cinematic runs without any graphical issue	Cinematic runs with minor graphical issues on the water object. High waves show underwater textures and cover partially the character.	FAIL

Test Case #4

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check Parkour System functionality with mid-range Graphics Card	Check functionality in Training level	Parkour System works smooth	Parkour System works smooth	PASS
	Check functionality in Mission level with Post-Processing	Parkour System works smooth	The character often ends up under the texture.	FAIL
	Check functionality in Mission level without Post-Processing	Parkour System works smooth	Parkour System works smooth	PASS

Test Case #5

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check Parkour System functionality with high-end Graphics Card	Check functionality in Training level	Parkour System works smooth	Parkour System works smooth	PASS
	Check functionality in Mission level with Post-Processing	Parkour System works smooth	Parkour System works smooth	PASS
	Check functionality in Mission level without Post-Processing	Parkour System works smooth	Parkour System works smooth	PASS

Test Case #6

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check Climbing System functionality with mid-range Graphics Card	Check functionality in Training level	Climbing System works smooth	The Climbing System works smoothly with very occasional bugs	PASS
	Check functionality in Mission level with Post-Processing	Climbing System works smooth	The character often ends up under the texture.	FAIL
	Check functionality in Mission level without Post-Processing	Climbing System works smooth	The Climbing System works smoothly with very occasional bugs	PASS

Test Case #7

Test scenario	Test case	Expected result	Actual result	Pass/Fail
Check Climbing System functionality with high-end Graphics Card	Check functionality in Training level	Climbing System works smooth	The Climbing System works smoothly with very occasional bugs	PASS
	Check functionality in Mission level with Post-Processing	Climbing System works smooth	The Climbing System works smoothly with very occasional bugs	PASS
	Check functionality in Mission level without Post-Processing	Climbing System works smooth	The Climbing System works smoothly with very occasional bugs	PASS

Bugs

Below there is a table with a list of the bugs found in the different test cases.

The bug report table contains three identified issues. The first bug, identified by bug ID 001, describes minor graphical issues that affect the cinematic experience of the game, particularly with the presence of high waves that partially cover the character and show underwater textures. Although this issue is marked as low severity, it is still open and requires attention from the developer.

Bug ID 002, on the other hand, has a critical severity rating and has been reported by a player or QA tester, indicating that it affects the game's playability. The issue involves the character ending up under the texture, which can be frustrating and may lead to a poor gaming experience. The bug is still open and assigned to Daniel Bellido for resolution.

Finally, bug ID 003 has a medium severity rating and involves occasional bugs with the Climbing System of the game, particularly with a mismatched animation. This issue has been reported by the developer and is still open. The bug was reported on 5th March 2023, and it has not yet been resolved.

In summary, the bug report table provides an overview of the identified issues in the project, including a brief description of each issue, its severity rating, status, and the individuals responsible for reporting and resolving them.

Bug ID	Bug Description	Bug Severity	Bug Status	Bug Reporter	Bug Assigned to	Date Reported	Date Resolved
001	Cinematic runs with minor graphical issues High waves show underwater textures and cover partially the character.	Low	Open	Project Supervisor	Daniel Bellido	02/04/2023	-
002	The character often ends up under the texture.	Critical	Open	Player/QA Tester	Daniel Bellido	25/04/2023	-
003	The Climbing System works with very occasional bugs. Target missmatched	Medium	Open	Developer	Daniel Bellido	05/03/2023	-

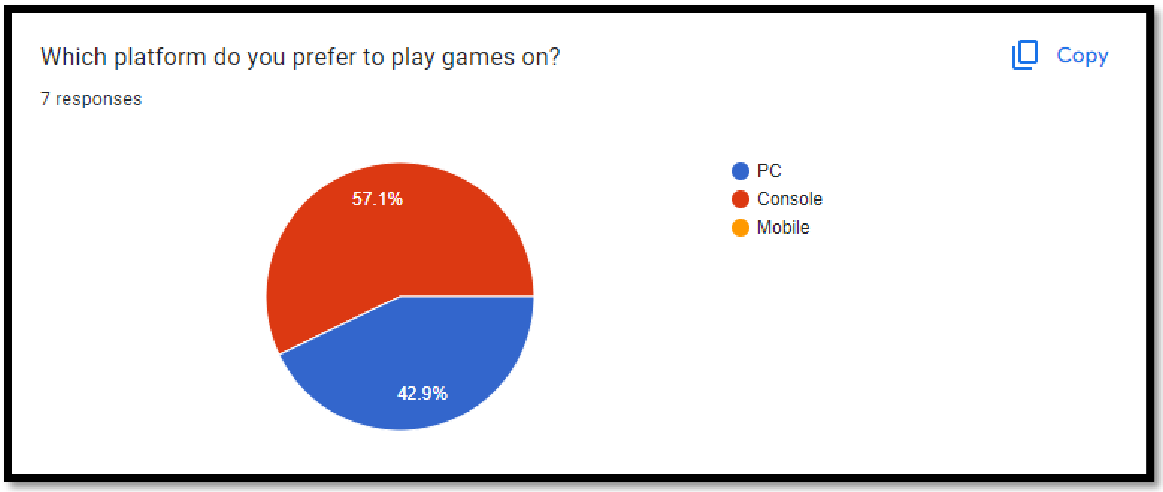
User Feedback

Obtaining feedback from players during the testing of a game is a critical step in the development process. It helps identify bugs and issues that may have been overlooked by developers, as well as provides insight into how players interact with the game. This feedback can then be used to improve the game and enhance the overall user experience. Additionally, involving players in the testing process can also help build a community around the game and increase its popularity. Overall, player feedback is a valuable resource for game developers and should be incorporated into the testing process whenever possible.

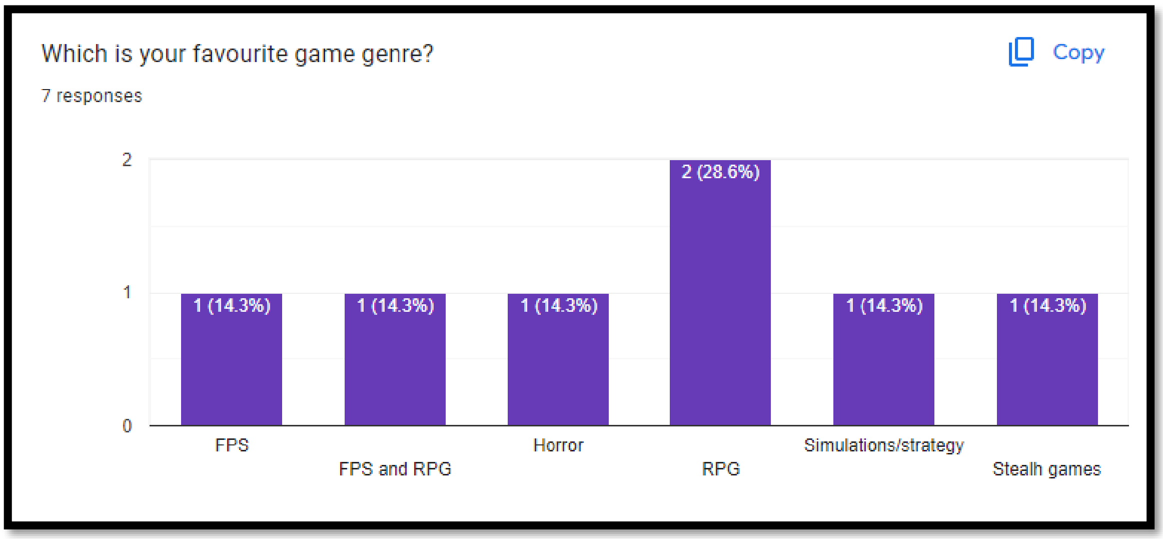
For this project, the players wanted to get involved in the testing phase so that they could offer another point of view and report any problems found in the game.

For this reason, a brief survey has been created that includes various questions about their tastes, information about their devices, and personal experience with the prototype. Unfortunately, the participation has not been very high but enough to confirm certain problems with the project. The survey consists of 14 questions and a total result of 7 answers has been obtained.

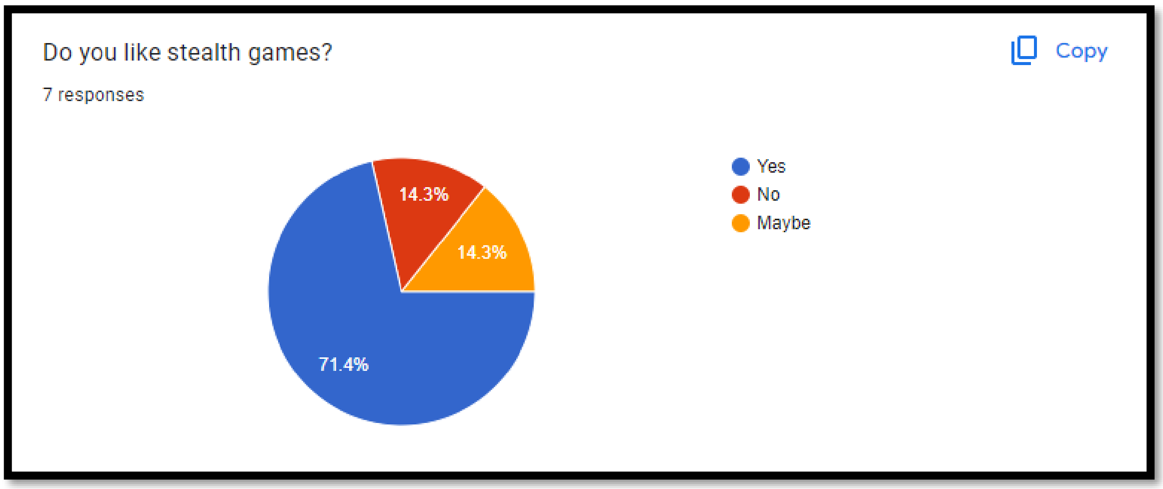
The first question in the survey was to find out a little about the audience and their preferences. For this reason, the first thing asked was what their preferred platform for playing video games was. And as can be seen in the figure, 57.1% of gamers prefer to play on video consoles compared to 42.9% who prefer to play on a PC.



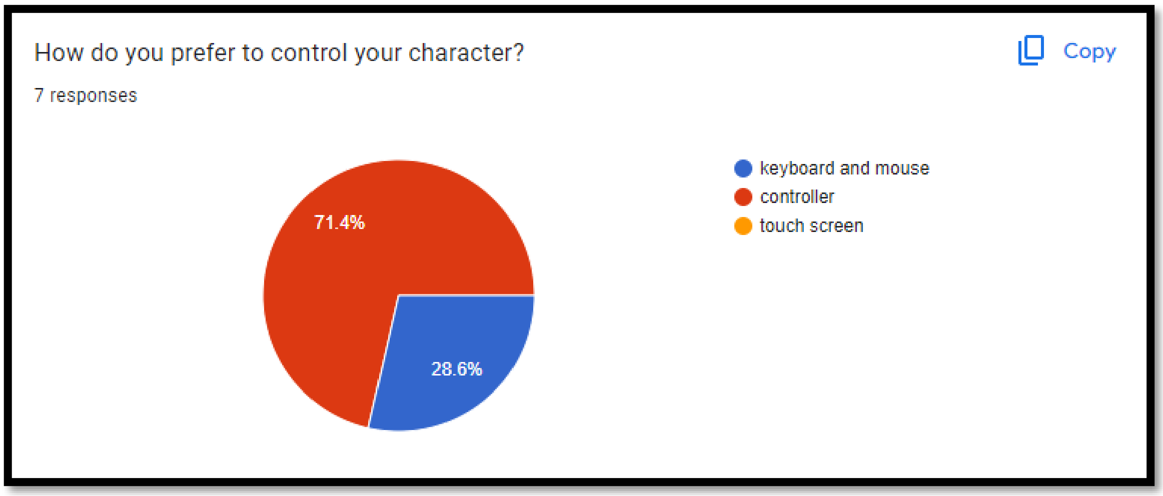
The second question asked about the type of genre they preferred to play and here, despite being a small group of people, the answers were very diverse. Carefully analyzing the graph we can conclude that the RPG is the most loved genre, followed by the FPS, while other genres such as horror, strategy, and stealth are in the minority compared to those previously mentioned.



The third question in the survey was already beginning to address issues more related to the project and that is that since it is a stealth game, it seemed interesting to ask if the stealth genre was well received. The response from the users was quite positive since 71.4% of those surveyed answered yes to whether they liked the stealth genre, while 14.3% answered no and the other 14.3% answered maybe.

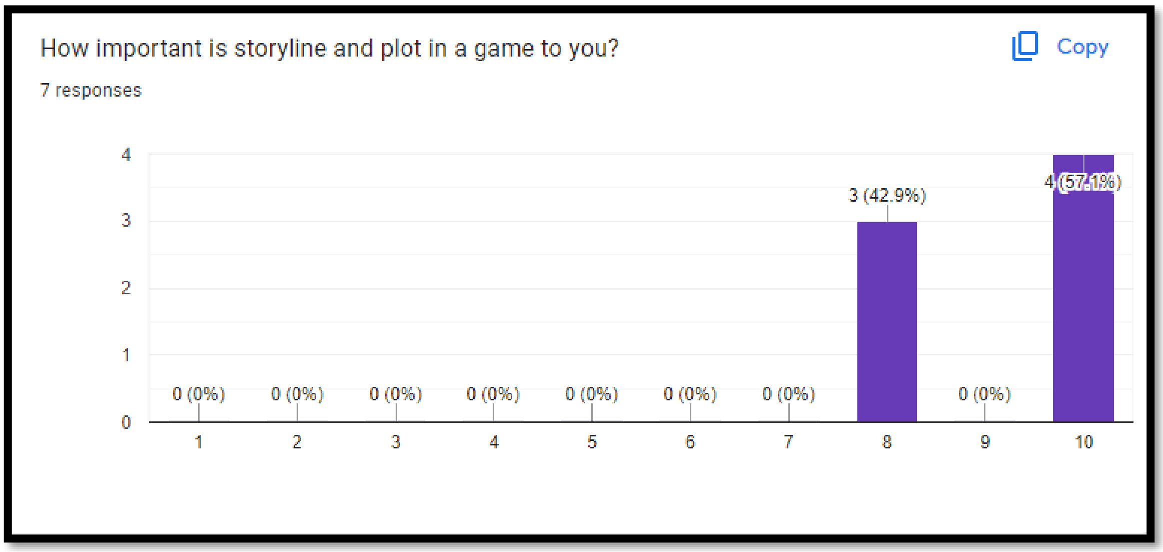


The next question is intended to find out what type of device players prefer to use to play games as this is an important element for many gamers and something to consider when developing games. 71.4% responded that they prefer to use a gamepad while 28.6% are more inclined to use a keyboard and mouse.



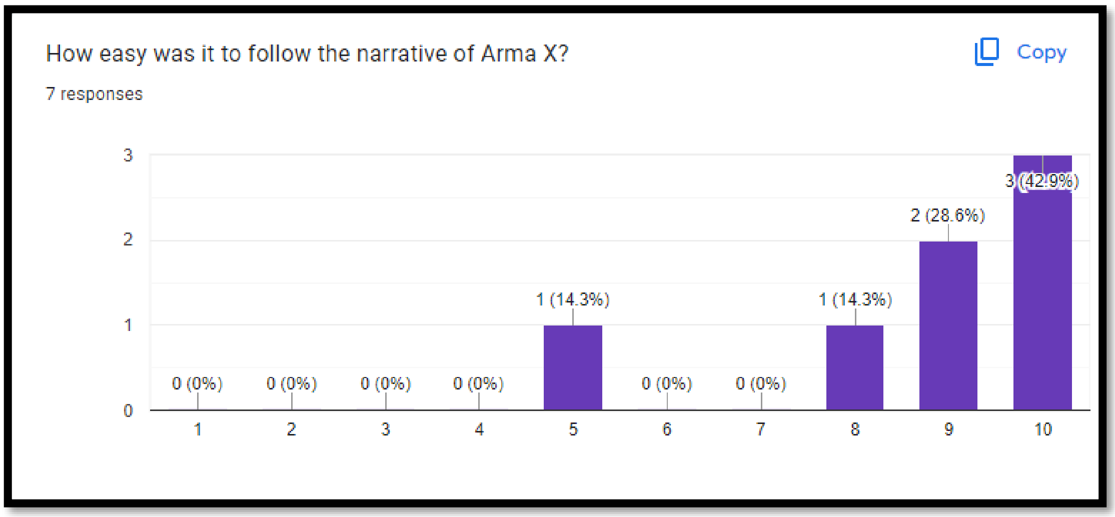
The fifth question introduces the narrative element and asks if players are interested in this type of approach. Responses are on a scale of 0 to 10 where 0 means that the narrative is not important while 10 mean that the narrative in a video game is very important.

In this sense, the vast majority, 57.1%, consider that the narrative in a video game is very important, while 42.9% of those surveyed consider that the narrative is simply important.



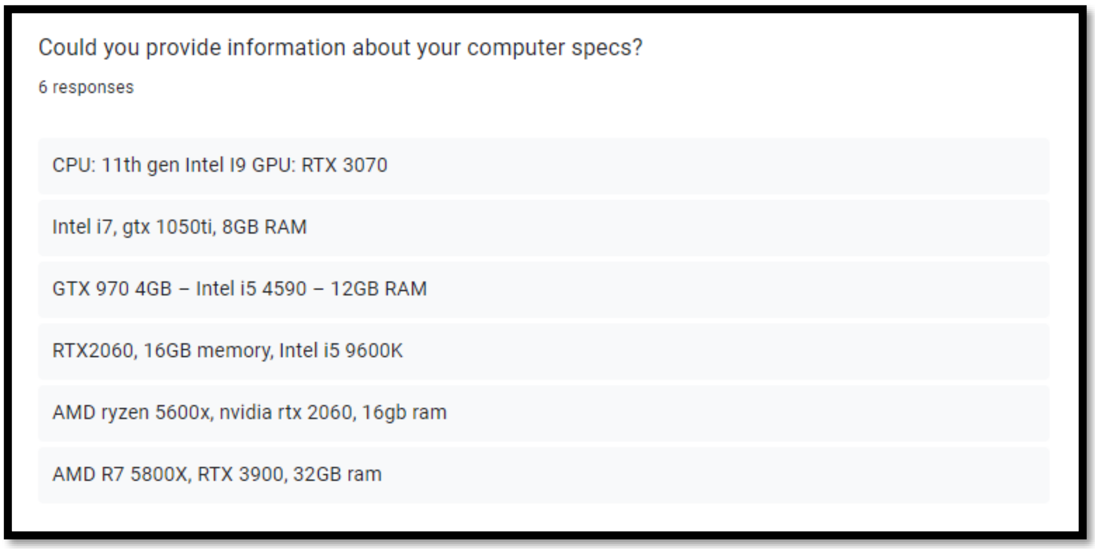
As for the narrative proposed by Arma X, users are asked how easy is to follow the narrative of the prototype, where 10 means it has been easy to follow and 0 means very difficult.

42.9% of those surveyed have answered that the narrative has been very easy to follow, followed by 28.6% who have responded with a 9 out of 10, another user considers that it has been easy and another has rated this question with a 5 out of 10.



After knowing the interest that the respondents have for the narrative, it is time to get into issues more related to the technical aspect. For this reason, users are asked to share information about their computer specs in order to assess the impact it may have on game performance on their devices.

Of the 6 answers obtained, it can be seen how there is a technical tie in terms of graphics cards since two of them could be considered low-end, another two mid-range, and the RTX3070 and RTX3090 are crowned as the only high-end graphics cards. Therefore, it is expected that many of the respondents will experience some kind of problem with the prototype.


Could you provide information about your computer specs?

6 responses

CPU: 11th gen Intel I9 GPU: RTX 3070

Intel i7, gtx 1050ti, 8GB RAM

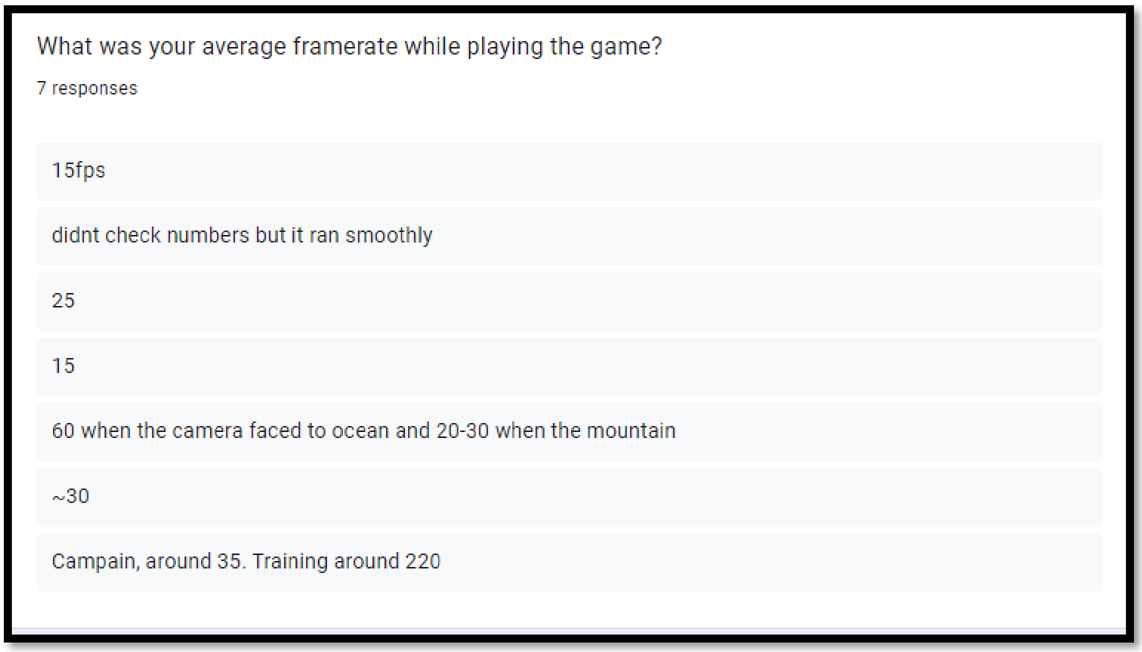
GTX 970 4GB – Intel i5 4590 – 12GB RAM

RTX2060, 16GB memory, Intel i5 9600K

AMD ryzen 5600x, nvidia rtx 2060, 16gb ram

AMD R7 5800X, RTX 3900, 32GB ram

The next question asks players what frame rates they have experienced with the prototype. The answers show that only two players have been able to enjoy the game stably while the rest have suffered a frame rate of less than 30 reaching the minimum of 15. It can be deduced that the only people who have been able to enjoy the game are those who have the RTX30 on their computers.



What was your average framerate while playing the game?

7 responses

15fps

didnt check numbers but it ran smoothly

25

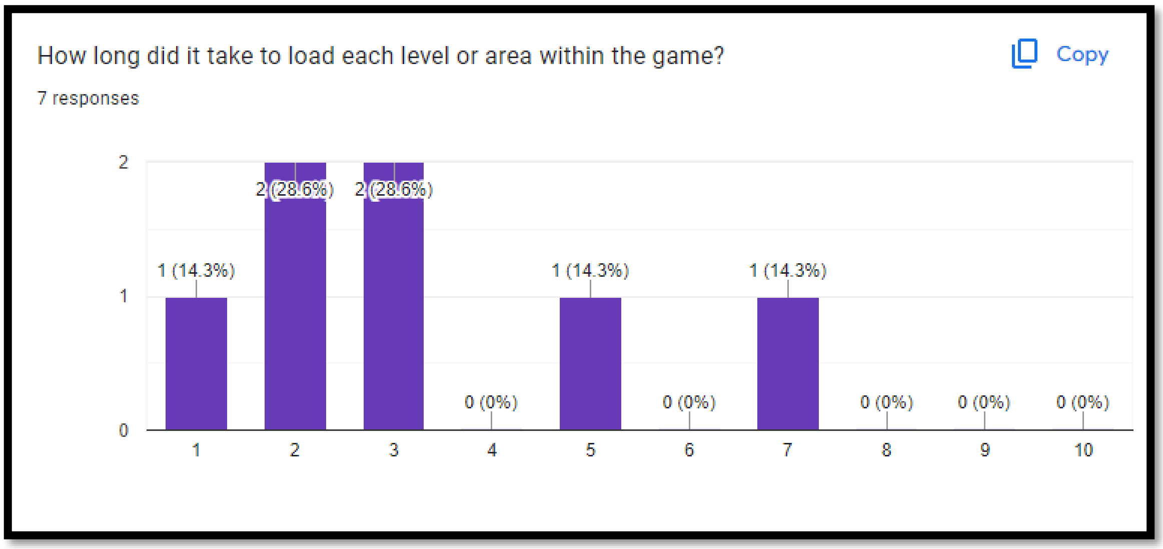
15

60 when the camera faced to ocean and 20-30 when the mountain

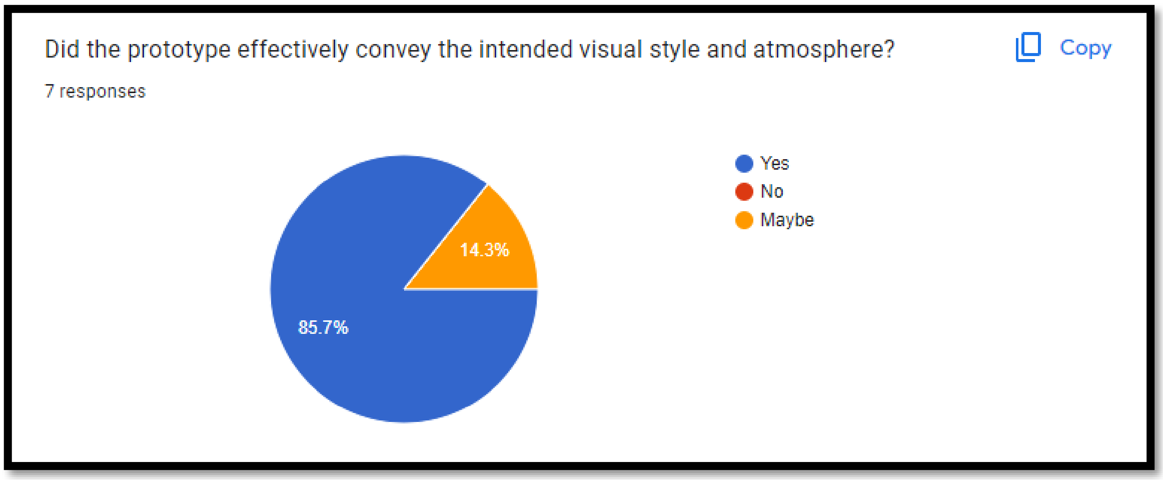
~30

Campain, around 35. Training around 220

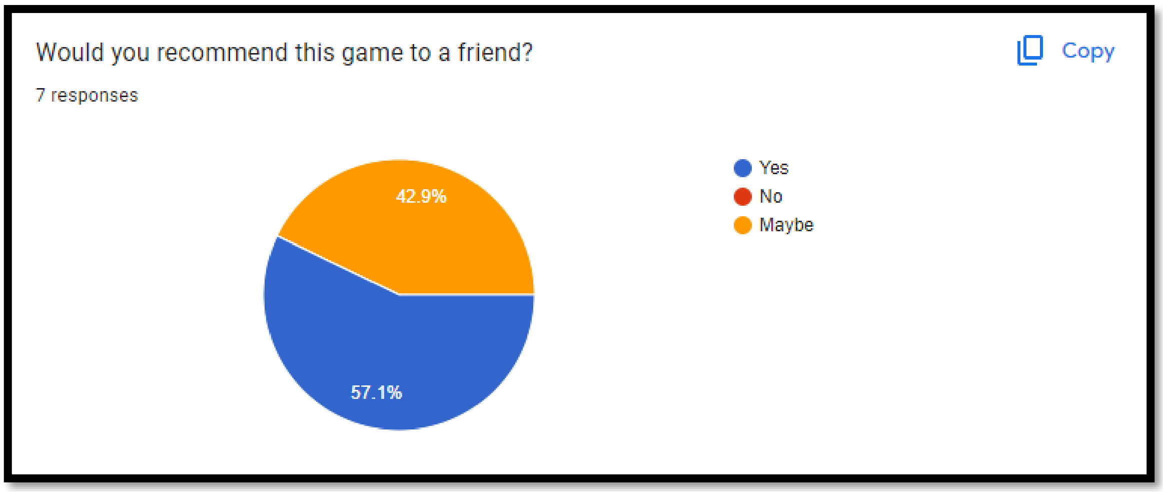
Given the optimization problems and the large number of objects at the main mission level, a study was conducted to evaluate the loading sensation experienced by users. Participants were asked to rate the loading time of the game on a scale of 1 to 10, where a rating of 1 indicated a fast load time and 10 indicated a very slow load time. The findings indicate that most users reported a relatively fast loading time, while a minority of users reported experiencing some difficulty with loading.



The survey included a question about the compatibility of the visual content of the prototype with the style of the story it conveys. The results indicate that 85.7% of the participants confirmed the compatibility while the rest of the respondents indicated their uncertainty regarding the compatibility.



Another inquiry aimed to determine the participants' willingness to recommend the game to their acquaintances despite the prototype's drawbacks. The findings show that 57.1% of the participants were willing to recommend the game. However, 42.9% of the respondents expressed their hesitance to recommend the game, indicating that they would only recommend it if the prototype was more polished.



Evaluation

As can be seen in the different cases exposed, the tests have been carried out with different equipment to check if the bug was caused by a software error or a lack of performance by the hardware. In this series of cases, some bugs are repeated systematically in mid-range equipment.

This bug started right after I implemented the volume post-processing, so this was initially suspected as the cause of the problem. After several tests, code, and component reviews, the post-processing volume was disabled to confirm if this was what caused the character to stop colliding with the terrain and fall into the void.

Later, it was tested again in another scene and after checking that there were no errors, it was concluded that the post-processing was undoubtedly the cause of the problem.

From that moment on, the question was whether this problem was replicated in equipment with better performance, so these same test cases were carried out at Kingston University where the equipment has high-end graphics cards. There the results were surprisingly good and the game ran at a stable 60fps at all times and it seemed that the game did not suffer from any kind of lack of optimization.

The next step was to rule out that the problem only occurred in the private computer where the project was regularly developed. For this reason, a build of the project was made available to some users so that they could test the game and could give me technical data about their devices.

The results of the survey showed how only a third of the users were able to run the game without problems while the rest of the players had the expected critical bug caused by poor level optimization and post-processing resources.

In this sense, it is indisputable that the project needed exhaustive optimization work and to implement the graphics adjustment functionality so that low-performance devices can run and enjoy the game without problems.

To summarize, the analysis shows that the project has performance issues on modest computers, indicating a need for further investigation and optimization. The use of profiler tools to identify spikes that impact game performance is recommended, along with optimizing every aspect of the project. Additionally, the implementation of graphics adjustments as additional options to the game is necessary, given the varied components of players' computers as shown in the survey results. Customizing the game to meet the needs of each device is crucial to enhancing the game experience.

7. Legal, Social, Security and Ethical Issues

PEGI

The game is intended to present explicit violence against human characters, such as shooting them, stealthily executing them by suffocation, or the use of explosive weaponry such as grenades, being run over by vehicles or simply being beaten to death.

According to the description of the PEGI content, the game would fall into the PEGI 16 or PEGI 18 rating. The game could also present some degree of bad language which is allowed in both PEGI 16 and PEGI 18 but the element that rules it out to enter the PEGI 16 category and definitively grant it a PEGI 18 category would be the inclusion of political elements and symbols that promote hatred, racism, and homophobia, due to the historical background through which the narrative of the game takes place.

18
www.pegi.info
VIOLENCE
@*!
BAD LANGUAGE
DISCRIMINATION

Copyright

To streamline the development process, the project will use third-party assets purchased from the official Unity store. In accordance with Unity policies, once you have purchased an asset from the official assets store, the asset becomes your property and therefore you can market your game royalty-free.

Regarding the use of the graphics engine, Unity allows you to sell any product developed with the personal edition of the engine at no additional cost unless the product generates revenues greater than $100,000 in a 12-month period.

The use of animations downloaded from the Mixamo website, Adobe allows the use of its assets in commercial products such as video games. What is totally unauthorized is the resale of directly downloaded Mixamo assets and packages.

In order to use models and animations developed in Maya, the use of the student version for commercial purposes is strictly prohibited. In order to trade assets produced with Maya, a paid license is required.

Ethical Issues

Due to the nature of the events that occurred during the 20th century, it is understandable that some people may experience rejection for the content related to

Nazism shown in the video game, even that it could be banned in some countries such as Germany, although Germany no longer It has censored the use of symbols of unconstitutional organizations since 2018 and has removed old games such as the Wolfenstein saga, Mortyr, and Commandos from its List of Harmful Media.

Although the game is not expected to be extremely violent, additional problems can be generated by the violence that the game may contain. In this aspect, the case is much more diffuse since some countries have different regulations on the violence exposed in video games. Some of these countries are Venezuela, Brazil, China, Japan, South Korea, Australia, Malaysia, Singapore, Germany, United Kingdom, Saudi Arabia, United Arab Emirates, Iran or Pakistan, which have even censored or changed some games to suit their markets.

Data Security

The game will not be an online game and will not require any type of personal data, so no data security policy is applied since they are totally unnecessary.

8. Critical Review and Conclusion

Achievements

This project has produced a comprehensive collection of individual accomplishments spanning multiple domains, encompassing technical and design aspects. It is noteworthy to highlight that the mere act of conceptualizing and developing a game of this magnitude represents a significant accomplishment, as it poses a formidable challenge. This project is particularly remarkable since it marks my first instance undertaking a project of such scale. While there remains room for further refinement, I derive a reasonable degree of personal fulfilment from the current state of the project.

One of the achievements in my Final Year Project is the enhancement of my skills in the area of gameplay programming, which involved implementing a sophisticated movement system that could accommodate a substantial number of animations, as well as developing mechanics such as the Parkour System and the Climbing System. Furthermore, this skill development facilitated the conceptualization of a more intricate level design that incorporates an interactive environment replete with objects and interactive components. Substantial visual enhancements were also implemented, including particle effects and several post-processing volumes that contributed to the creation of more realistic lighting in the game.

I am highly content with the opportunity to introduce a narrative design that significantly elevates the allure of the project. This component encompasses the integration of a linear story, which is accompanied by an introductory video upon launching the application, as well as a cinematic opening sequence with dialogues at the beginning of the game. Moreover, the inclusion of such narrative elements has facilitated the creation of a Main Menu and loading screen that prioritizes user experience (UX) design principles.

In short, at the Game Design level, quality documentation development has been achieved with a Game Design Document, reports, and even online devlogs that help to understand the work. Without a doubt, the Final Year Project has been a very valuable experience for future personal and professional work.

Areas to improve

Even so, the project has had many problems and requires a lot of research and a lot of work ahead, as it has been possible to demonstrate in the testing phase carried out by some classmates. The bad performance of the game on modest computers marks a turning point in this project and now I have put all my attention to solving this problem since the optimization of the project is vital to have a good product. Therefore, the game must enter an optimization phase before continuing to implement features, since if it is left until the end to optimize the game with all the implementation, it will become an impossible task to finish. After all, this project is developed with an Agile approach and requires these iterations for the project to progress.

Another important aspect that could be much better implemented and will undergo an overhaul is the design and technical implementation of the main character. At the moment, the numbers of scripts that the character controls are few, and the PlayerMovement script is in charge of carrying out tasks that go beyond what was initially planned. This script not only controls the movement of the player but also controls different player states and a large number of animations. Continuing to implement features with this design becomes cumbersome and often presents errors that have to be constantly corrected, which represents a large investment of time invested in a very inefficient way.

A much more practical, maintainable, and scalable solution would be to create a custom state machine for the main character that would be able to handle different stances and animations, as well as different types of movement depending on the state it is in. An example of two different states of movement would be the default that allows a free camera to rotate around the player and another aimed state where the camera would be fixed on the back of the player and the player should use another type of movement similar to that of the First Person Shooter. However, all the movement code, including the Parkour System and the Climbing System, has a dependency on a single script and this poses many complications and errors when trying to add new functionality.

Time Management

The experience of having worked throughout the course with this project has had a positive impact in many aspects on a personal level; however, there have also been times in which the project has suffered major development stoppages due to other tasks unrelated to this project and this has largely frustrated the course of development.

Initially, the project had a series of objectives and tasks that were perfectly planned, even allowing extra time for errors and complications. What were not foreseen are that other course modules would be as demanding and long as, for example, the parallel programming project and animation and modelling assignments.

Also, there were some other issues with other team projects that became a waste of time. In short, I invested more than a month in group project for the narrative assignment and a month later part of the team wanted to restart the project with another idea that involved discarding all my work done, which forced me to leave the group and transfer the narrative project to my Final Year Project.

The requirement to implement narrative and cut-scenes in the project was foreseen but it was not a priority and due to the problem in the narrative assignment this requirement became a very high priority since the deadline was very close and I had wasted too much time with the team project. Therefore it was difficult to incorporate this new requirement without altering the original plan which implied implementing features ahead of time in a very haphazard way, which affected and frustrated the implementation of other requirements such as Artificial Intelligence.

Even so, very hard work has been done to get to this current state, and the best that has been done with the little time that has been had although I am very frustrated that I have not been able to implement a basic Artificial Intelligence, a simple combat system and a questline that allows gamers to enjoy a playable experience.

Personal Learning

Undoubtedly this project has helped me to learn a lot about various topics and to better understand how to assemble all these various topics in a single project, giving me a better insight into Game Design and Game Development.

This experience has given me the opportunity to work in much more detail on documentation and the field of design, something that, in addition to enriching my knowledge, has made me have a great time.

I have also been able to see more closely and understand how important the role of the Project Manager is and the need to plan the procedure with the help of tools such as Trello, Gantt Chart, Git, and the importance of having a blog that communicates the progress of the project.

I have also learned a lot about testing and its importance in game development, not only for one person but also for projects carried out by a team. Although my knowledge in this area has been very basic, I have been able to see with my own eyes the importance of designing test cases, having a bug report, and, above all, having user feedback.

On a technical level, I gained much more knowledge about the Unity engine and many of its features such as Mecanim, Timeline, Cinemachine, ProBuilder, Particle Effects, Shader Graph, different render pipelines, and the use of different classes such as the Scriptable Object, which opened up a new world of possibilities for me when I saw that I could customize the Unity editor and create more personalized projects. In short, I was able to put into practice the knowledge acquired throughout all these years at the university and this project has motivated me to continue learning about more advanced topics, especially with the development of tools for the engine and about new ways of programming in Unity.

Future Work

After having run out of time at the university and from the current state of the project, it is evident that the game needs to be worked on in many aspects and that further research needs to be done.

First of all, the project urgently needs an optimization process that debugs all the performance problems. To do this, the Unity profiler tool will be used and an attempt will be made to identify if, in addition to the volume of post-processing, other factors slow down the game. Once the project has a decent performance of about 60 frames per second, basic artificial intelligence based on a simple Finite State Machine will be implemented.

After that, it will go to a testing phase with a greater scope where more participants will be tried to receive more feedback. Once the feedback is received, the project will be re-planned, taking into account the feedback from the users, while at the same time, research will be carried out on different topics that help the progress of the project.

These investigations to be carried out consist of taking a series of online courses instructed by the developer Penny de Byl, who has a long history in the development of video games with Unity. Some of these courses focus on Artificial Intelligence and cover topics like Goal Oriented Action Planning and Behaviour Trees while other courses focus on optimization with topics like Entity Component System (ECS).

Once the knowledge has been expanded and an appropriate plan has been developed, the main character's code will be refactored, implementing a new system based on State Machine that provides an optimal, maintainable, and easy-to-scale implementation. Once this objective has been achieved, the Combat System will be addressed, which will interact with the basic AI already implemented to test and explore all the possibilities.

The next step would be to improve the level design, and implement a basic questline with some more animations so that the game can be minimally playable.

Finally, a massive testing phase will be entered again, trying to have a wide reach within the online communities so that they can test the game and can provide the necessary feedback to analyze again the needs of the project.

9. Research and References

1. *15 countries that ban video games* (2016) *Expert blog for professionals in the video game industry*. Available at: https://codeswholesale.com/blog/15-countries-that-ban-video-gamesin-different-countries/ (Accessed: 20 October 2022).
2. *All about: PEGI age ratings* (no date) *Askaboutgames.com*. Available at: https://www.askaboutgames.com/pegi-age-ratings (Accessed: 20 October 2022).
3. American Experience | PBS (2011) *FDR and policing the world: Hitler's threat*. Youtube. Available at: https://www.youtube.com/watch?v=FtDxjVCu56E (Accessed: 10 March 2023).
4. Ander, A. C. (2023) *Como hacer efecto de voz de radio en Audacity*. Youtube. Available at: https://www.youtube.com/watch?v=CAqSZDAWjLE (Accessed: 28 March 2023).
5. Aura, C. G. (2020) *Cutscene in Unity 3D | Timeline in Unity | CG Aura*. Youtube. Available at: https://www.youtube.com/watch?v=w6lc8svzBms (Accessed: 28 March 2023).
6. Aversa, D. (2022) *Unity Artificial Intelligence Programming: Add powerful, believable, and fun AI entities in your game with the power of Unity*. 5th edn. Birmingham: Packt Publishing.
7. BBC News (2018) 'Germany lifts total ban on Nazi symbols in video games', *BBC*, 10 August. Available at: https://www.bbc.co.uk/news/world-europe-45142651 (Accessed: 20 October 2022).
8. Brackeys (2017a) *How to make a LOADING BAR in Unity*. Youtube. Available at: https://www.youtube.com/watch?v=YMj2qPq9CP8 (Accessed: 1 April 2023).
9. Brackeys (2017b) *Introduction to AUDIO in Unity*. Youtube. Available at: https://www.youtube.com/watch?v=6OT43pvUyfY (Accessed: 23 March 2023).
10. Brackeys (2017c) *START MENU in Unity*. Youtube. Available at: https://www.youtube.com/watch?v=zc8ac_qUXQY (Accessed: 24 March 2023).
11. British Pathé (2011) *Hindenburg disaster: Real zeppelin explosion footage (1937) | British pathé*. Youtube. Available at: https://www.youtube.com/watch?v=CgWHbpMVQ1U (Accessed: 10 March 2023).
12. British Pathé (2014a) *Hitler returns to Germany from France (1940) | British pathé*. Youtube. Available at: https://www.youtube.com/watch?v=g3xRVKkvx9A (Accessed: 10 March 2023).

13. British Pathé (2014b) *Nuclear (1970-1979)*. Youtube. Available at: https://www.youtube.com/watch?v=GUPf9z_QcsQ (Accessed: 10 March 2023).
14. British Pathé (2014c) *Selected Originals - Mao In Moscow (1950)*. Youtube. Available at: https://www.youtube.com/watch?v=gFLz2juVbY8 (Accessed: 10 March 2023).
15. de Byl, P. (no date) *Advanced AI for games with goal-oriented action planning*, *Learn @ Holistic3D*. Available at: https://www.h3dlearn.com/course/advanced-ai-for-games-with-goal-oriented-action-planning (Accessed: 30 April 2023).
16. *Can I use assets made in student copy after purchasing maya LT for commercial* (2019) *Autodesk Community*. Available at: https://forums.autodesk.com/t5/maya-forum/can-i-use-assets-made-in-student-copy-after-purchasing-maya-lt/td-p/8889390 (Accessed: 19 October 2022).
17. Carter, R. *et al.* (2013) *Unity 4.X game AI programming*. Birmingham: Packt Publishing.
18. Cuchovasky, A. (2014) *La Segunda Guerra Mundial en Color (Documental completo en español)*. Youtube. Available at: https://www.youtube.com/watch?v=jqlSzP1_CEA (Accessed: 10 March 2023).
19. Dean, J. (2023) *Unity character animation with mecanim*. Birmingham: Packt Publishing.
20. Discovery, U. K. (2019) *How Hitler invaded half of Europe | greatest events of World War 2 in colour*. Youtube. Available at: https://www.youtube.com/watch?v=rXt8rU97NMQ (Accessed: 10 March 2023).
21. Download Archive and Beta Program (no date a) *Dystopia Soldier*, *Unity.com*. Available at: https://assetstore.unity.com/packages/3d/characters/humanoids/dystopia-soldier-180489 (Accessed: 7 December 2022).
22. Download Archive and Beta Program (no date b) *Flooded Grounds*, *Unity.com*. Available at: https://assetstore.unity.com/packages/3d/environments/flooded-grounds-48529 (Accessed: 15 March 2023).
23. Download Archive and Beta Program (no date c) *HQ Hangar Free*, *Unity.com*. Available at: https://assetstore.unity.com/packages/3d/environments/hq-hangar-free-212795 (Accessed: 15 February 2023).
24. Download Archive and Beta Program (no date d) *Military Outpost Set*, *Unity.com*. Available at: https://assetstore.unity.com/packages/3d/environments/industrial/military-outpost-set-91227 (Accessed: 15 January 2023).
25. Download Archive and Beta Program (no date e) *Modern Forward Military Base Kit*, *Unity.com*. Available at: https://assetstore.unity.com/packages/3d/environments/modern-forward-military-base-kit-145564 (Accessed: 15 January 2023).

26. Download Archive and Beta Program (no date f) *Old Rowboat*, *Unity.com*. Available at: https://assetstore.unity.com/packages/3d/vehicles/sea/old-rowboat-31917 (Accessed: 15 March 2023).
27. Download Archive and Beta Program (no date g) *Simple Water Shader HDRP*, *Unity.com*. Available at: https://assetstore.unity.com/packages/2d/textures-materials/water/simple-water-shader-hdrp-207454 (Accessed: 15 February 2023).
28. Ecuador, C. (2014) *La batalla de Stalingrado*. Youtube. Available at: https://www.youtube.com/watch?v=i0qu5g2MW6Q (Accessed: 10 March 2023).
29. Fermi, W. (2015) *LOS AVANCES ARMAMENTÍSTICOS EN LA ALEMANIA NAZI*. Youtube. Available at: https://www.youtube.com/watch?v=xQAtU60sSP0 (Accessed: 10 March 2023).
30. Free Documentary-History (2022) *The Rise of Adolf Hitler | Germany's Fatal Attraction: Part 1 | free documentary history*. Youtube. Available at: https://www.youtube.com/watch?v=0enHZd8H0XA (Accessed: 10 March 2023).
31. Game Maker's Toolkit (2017) *What makes good AI?* Youtube. Available at: https://www.youtube.com/watch?v=9bbhJi0NBkk (Accessed: 1 May 2023).
32. Game Maker's Toolkit (2020a) *How do stealth games deal with detection? - School of stealth part 3*. Youtube. Available at: https://www.youtube.com/watch?v=uF6c8KJuuEk (Accessed: 1 May 2023).
33. Game Maker's Toolkit (2020b) *How stealth game guards see and hear - School of stealth part 1*. Youtube. Available at: https://www.youtube.com/watch?v=Ay-5g360Ffc (Accessed: 1 May 2023).
34. Game Maker's Toolkit (2020c) *The five types of stealth game gadget - School of stealth part 2*. Youtube. Available at: https://www.youtube.com/watch?v=QLWC081dDpc (Accessed: 1 May 2023).
35. Hight, J. and Novak, J. (2008) *Game Development Essentials: Game project management*. Delmar Pub.
36. de la Historia, G. M. (2022) *El sábado negro - Bombardeando los muelles de Londres durante la Segunda Guerra Mundial*. Youtube. Available at: https://www.youtube.com/watch?v=6tnIhzts8Ns (Accessed: 10 March 2023).
37. Htwo (2022) *The most influential AI in video game history*. Youtube. Available at: https://www.youtube.com/watch?v=4hcdvvxuS2U (Accessed: 1 May 2023).
38. Keith, C. (2010) *Agile game development with scrum*. Boston, MA, USA: Addison-Wesley Educational.

39. KickFlip (2018) *Vietnam War - Part 1 [Real Footage]*. Youtube. Available at: https://www.youtube.com/watch?v=rrVqBqqq5AA (Accessed: 10 March 2023).
40. Kremers, R. (2009) *Level design: Concept, theory, and practice*. Natick, MA, USA: A K Peters.
41. Learn Programming and Math (2022) *Git Large File Storage LFS | GitHub blocks pushes that exceed 100 MB | solucionar error github LFS*. Youtube. Available at: https://www.youtube.com/watch?v=Uyvm4hLkEAQ (Accessed: 23 March 2023).
42. MikePro (2021) *Night vision goggles - sound effect - free MP3 download*. Youtube. Available at: https://www.youtube.com/watch?v=gpt11SD3snA (Accessed: 23 March 2023).
43. Millington, I. and Funge, J. (2009) *Artificial Intelligence for Games*. 2nd edn. Philadelphia, PA, USA: Focal Press.
44. *Mixamo character and animation use* (2021) *https://community.adobe.com*. Available at: https://community.adobe.com/t5/mixamo-discussions/mixamo-character-and-animation-use/m-p/12182711 (Accessed: 1 May 2023).
45. Nadeem, A. (2022) *How to make Car Lights and Flare Effect in Unity particle system*. Youtube. Available at: https://www.youtube.com/watch?v=Q_flwXeX3AQ (Accessed: 23 March 2023).
46. Nerd Head (2022) *Make underwater camera - realistic effect| #UnityIn60Sec*. Youtube. Available at: https://www.youtube.com/watch?v=9OYzeZvhTtM (Accessed: 28 March 2023).
47. nikobmxero (2007) *Explosiones de bombas atomicas - Parte 1*. Youtube. Available at: https://www.youtube.com/watch?v=0SSxdn20tYw (Accessed: 10 March 2023).
48. Palmer, S. W. (2015) *Stalin's Final Speech 1952 [Subtitled]*. Youtube. Available at: https://www.youtube.com/watch?v=3nMDjKtTigQ (Accessed: 10 March 2023).
49. Problems, I. (2014) *Kamikaze pilots WWII*. Youtube. Available at: https://www.youtube.com/watch?v=4mTECUWP0Hk (Accessed: 10 March 2023).
50. Rogers, S. (2014) *Level up! The guide to great video game design*. 2nd edn. Nashville, TN, USA: John Wiley & Sons.
51. Sanjurjo, F. R. (2015) *La bomba ato ́mica, 70 an□os HD*. Youtube. Available at: https://www.youtube.com/watch?v=6xAy_XAif1Q (Accessed: 10 March 2023).
52. SleepySounds (2015) *Deep Sea Soundscape – 9 hours of underwater ambience – Deep Ocean Sleep Sounds*. Youtube. Available at: https://www.youtube.com/watch?v=UjQxhOXco_k (Accessed: 29 March 2023).
53. Smith, M. and Queiroz, C. (2023) *Unity 5.x Cookbook*. Birmingham: Packt Publishing.

54. *The anatomy of a design document, part 1: Documentation guidelines for the game concept and proposal* (1999) *Game Developer*. Available at:

https://www.gamedeveloper.com/design/the-anatomy-of-a-design-document-part-1-documentation-guidelines-for-the-game-concept-and-proposal (Accessed: 1 May 2023).

55. Timeline-World History Documentaries (2020) *The origins of communist China's war with Taiwan | secrets of war | timeline*. Youtube. Available at: https://www.youtube.com/watch?v=6FHHT_O-Khk (Accessed: 10 March 2023).
56. Vegas, J. (2019) *How to start a cutscene from a trigger with c# in unity tutorial*. Youtube. Available at: https://www.youtube.com/watch?v=pru5sx_hqeE (Accessed: 28 March 2023).
57. Villafañe, A. M. (2013) *Batalla de Normandía 1944 ('Dia D' completo)*. Youtube. Available at: https://www.youtube.com/watch?v=jHzhtcJHbww (Accessed: 10 March 2023).
58. VOCEDITALIA (2013) *Discorso del Duce Benito Mussolini a Taranto, 7 settembre 1934*. Youtube. Available at: https://www.youtube.com/watch?v=mxSzUlX59eI (Accessed: 10 March 2023).
59. *What do the labels mean?* (no date) *Pegi Public Site*. Available at: https://pegi.info/what-do-the-labels-mean (Accessed: 1 May 2023).
60. Wikipedia contributors (2022) *Game design document*, *Wikipedia, The Free Encyclopedia*. Available at: https://en.wikipedia.org/w/index.php?title=Game_design_document&oldid=1070154144.
61. (No date a) *Unity.com*. Available at: https://support.unity.com/hc/en-us/articles/205623589-Can-I-use-assets-from-the-Asset-Store-in-my-commercial-game (Accessed: 1 May 2023).
62. (No date b) *Unity.com*. Available at: https://support.unity.com/hc/en-us/articles/205253119-Can-I-make-a-commercial-game-with-Unity-Free-Personal-Edition (Accessed: 1 May 2023).