Final Year Project Interim Report

Full Unit – Interim Report

Building a Game

Prithvi Sathyamoorthy Veeran

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Julien Lange

Department of Computer Science

Royal Holloway, University of London

December 13, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 6943

Student Name: Prithvi Sathyamoorthy Veeran

Date of Submission: 13/12/2024

Signature: Prithvi Sathyamoorthy Veeran

Table of Contents

Abstract

With the game industry expanding rapidly over the last few years, the idea of games has drastically changed and adapted. With so many genres and styles that have been created, starting from the first pong game to the huge immersive world of rockstars Red Dead Redemption 2, building a game can take many different forms. Many aim to innovate and change the industry for good; some build on pre-existing formulas that are proven to be a staple within the audience; however, the idea of thinking outside box and pushing the limits of creativity and the current technology do not keep games bound to set ideologies, innovation is the main driving factor behind this all, and such an approach in this is exactly what I am aiming to achieve. This combination of ideas is exactly what I intend to take, with aims to design a 3-d top-down dungeon crawler game, a genre that is characterised by its exploration element of a procedurally generated map, with real-time combat and loot collection. Drawing inspiration from existing games such as the Diablo series and the Binding of Isaac. The game is set in a fantasy world where you, as the player, are tasked with navigating the dungeon, facing enemies, solving puzzles, and collecting items. This has the potential to demonstrate strong AI design, level generation, and player feedback.

The Godot engine has been selected for this game due to its robust open-source nature and powerful development capabilities. Godot's node-based architecture and GDScript (similar to the Python language) provide an accessible but powerful environment for rapid prototyping and feature development. Also, it has great support for procedural generation, and AI behaviours make it a great engine to use. Procedural generation has been an extensively studied topic in game design as a method for creating the element of replay ability effectively [12], and this project will be building upon established algorithms such as cellular automata and random walks [11]. Furthermore, Godot's lightweight framework allows for the game to run smoothly and efficiently, even on lower-end hardware, which will broaden the potential audience. The documentation and community tutorials [5] will be vital for this project, as they provide extensive resources that will help to resolve potential challenges encountered during development.

Some of the key functionalities of the game will include procedurally generated content. This will allow for endless combinations of map layouts and enemy encounters. This feature has become very popular in the modern game development scene, offering the ability to extend the gameplay without having to manually design and add each level [12]. Combat and exploration are main features to be seen as well. Combat will take place in real-time, with the player having to control their unique character through the dungeon, fighting enemies, using different abilities to do so. Drawing from the Diablo series, the combat system will involve basic attacks, a way to dodge/block, and powerups, creating an engaging and strategic combat experience for the player. The health system, item collection, and the possibility of traps add layers of difficulty to the game, along with the added puzzles along the way to create further engagement and critical thinking of the player.

The AI for enemies will be another critical feature, requiring them to behave intelligently in coordination with the player. These things include simple enemy patrol paths, detecting player, and engaging in combat with them, while more complex AI types could feature teamwork, ranged attacks and ambush attacks.

While the game concept can be seen as ambitious, there will be several technical challenges that I will face, such as the procedural generation and the AI behaviour. Procedural generation requires that I carefully balance the random elements, such as enemy placement and room layouts - things that can be error-prone. This will be a good learning curve for myself to ensure each dungeon feels cohesive and fair to the player. Additionally, implementing an AI that reacts intelligently to the player is a significant challenge too. Enemies should pose a threat while also being fun to fight.

The main goal of this project is to create a game, that is repayable and successfully combines elements such as procedural content generation with real time combat and an intelligent AI system,

with a smart code base that is complex in many ways. The idea behind games is to let the player feel like they are part of the world they are experiencing and delivering that in a polished and engaging way is important.

Chapter 1: **Introduction**

1.1 Background

The origins of the dungeon crawler genre come from tabletop role-playing games such as Dungeons & Dragons, where players are tasked with traversing complex environments filled with enemies, treasures, and traps. The classification of a game as a roguelike is dependent upon various significant and minor factors [6]. Video games like Rogue (1980) and Diablo (1996) [2] significantly contributed to the genre's popularity by incorporating hack-and-slash combat mechanics and procedural level generation, which subsequently became hallmarks of the characteristics of the genre. Diablo 2 (2000) [3] further enhanced these elements, creating a paradigm for modern dungeon crawlers through its diverse character classes, skill trees, and procedurally generated environments. Games like Hades (2020) and Torchlight (2009) have reinvented the genre in recent years with their dynamic combat systems, stylised graphics, and intricate narratives. These advancements have demonstrated the genre's versatility and everlasting appeal, prompting creators to continually generate creative ideas.

1.2 Aims & Objectives

1.2.1 Aims

This project aims to create a roguelike, top-down dungeon crawler game that integrates modern gaming technology with the nostalgia of traditional dungeon crawlers. The game contains procedurally generated dungeons, tough real-time combat, and an immersive visual and audio experience. Engineered for replayability, it seeks to impart the exhilaration of fighting, the allure of exploration, and the gratification of discovering treasures and vanquishing foes. The project aims to maintain superior technical quality while ensuring the game is engaging for a broad audience through the use of advanced game development techniques and the integration of accessibility features.

1.2.2 Objectives

- **Procedural Level Generation**: Leverage robust procedural generating methods to provide diverse and unpredictable dungeon configurations. This entails creating diverse landscapes featuring distinct themes, configurations, and levels of difficulty to sustain player engagement. Including traps and puzzles that will appear in the dungeon for added immersion and player engagement. This will be achieved through:
    - Developing algorithms, such as cellular automata, that utilise randomisation and weighted probability to produce unique dungeon layouts.
- **Engaging Combat Mechanics**: Develop combat systems that enable players to utilise various powers, including melee, ranged, and magical attacks. Develop an adaptive artificial intelligence that responds to player actions and delivers a tough gaming experience. Steps to implement can include:
    - Establishing a modular combat system featuring interchangeable attack types,
    - Develop behavioural scripts for the enemy AI to react dynamically to the players strategies.
    - Iteratively testing and balancing combat mechanics to guarantee gameplay that is fair and enjoyable.

- **Inventory and loot**: Create and implement an intuitive item and inventory system enabling players to obtain, organise, and use items discovered within the dungeon.
    - Developing a graphical inventory system enabling players to examine and arrange their possessions.
    - Implementing item categorisation into categories such as consumables, equipment, and quest items.
    - Developing functionality for equipping, utilising, or discarding items.
    - Ensuring seamless integration of the inventory system into the gameplay loop.
- **User Interface Design**: Develop an aesthetically pleasing and user-centric interface that promotes intuitive navigation and improves the overall gameplay experience. This encompasses health bars, inventory management systems, and a potential mini map.
- **Puzzles and Traps:** Design and install captivating puzzles and traps to enhance the player's experience and introduce further challenges within the dungeon. This will encompass:
    - Developing complex puzzles that need logical reasoning, pattern identification, or time for resolution.
    - Incorporating traps like pressure plates, concealed spikes, and dynamic obstacles that challenge the player's reflexes and strategic thinking.
    - Ensure that puzzles and traps are procedurally integrated into dungeon layouts to allow for dynamic gameplay.

1.3 Project Motivation

The dungeon crawler genre has consistently inspired and fascinated me, driven by a lifelong enthusiasm and passion for gaming. This project represents a personal endeavour to integrate my passion for gaming with the technological expertise I have developed through my academic and professional experiences. Since childhood, I have been captivated by the complex mechanics, strategic dilemmas, and immersive environments presented by games. This project gives an opportunity to develop something that is uniquely my own, merging creativity and technology into a game of which I can take pride.

The choice to create a dungeon crawler arises from its significant potential for creativity and player involvement. Titles such as Diablo 2 and Hades exemplify the integration of exploration, combat, and narrative into a captivating experience. This project draws from these ideas, seeking to expand the limits through the integration of procedural generation, dynamic combat mechanisms, and captivating level designs featuring puzzles and traps. Furthermore, the dungeon crawler genre presents significant opportunities for innovation and player involvement. This project leverages the genre's strengths while addressing shortages in current games. This project employs procedural generation, distinguishing it from many games that utilise static levels, thus guaranteeing that each playthrough is unique and enhancing replayability.

This project additionally serves as an opportunity to demonstrate my progression as a programmer and my capacity to manage a complex, multifaceted project. Utilising modern technologies such as the Godot 4 engine, procedural algorithms, and agile development methodologies, I intend to make a game that embodies both my enthusiasm, my technical expertise, and my creativity. This project aims to actualise a personal vision and share it among fellow game enthusiasts.

1.4 Technological Choices

While developing a game, the technologies used are an important factor to consider as a developer, as this is what allows the game to have a strong foundation. Each choice is backed by the ability to achieve the project's objectives. The combination of these technologies ensures that the project can meet its ambitious goals of delivering a game that is dynamic and polished.

1.4.1 Game Engine: Godot 4

Godot 4 was selected as the primary development engine because of its versatility, lightweight structure, and comprehensive support for both 2D and 3D game creation. The integrated script editor and robust features facilitate rapid iteration and testing, rendering it suitable for developing complex systems like procedural level creation and dynamic combat mechanics. It also allows for easy deployment on multiple platforms, allowing publication of the game a lot easier. Furthermore, Godot's open-source framework facilitates customisation, while its strong developer community provides dependable support and resources. These elements render Godot 4 an excellent and cost-effective option for this project.

1.4.2 Programming Language: GDScript

GDScript, the native scripting language of Godot, is essential to this project. Its lightweight syntax has been precisely designed for game development, guaranteeing seamless integration with Godot's extensive API. This close integration facilitates the efficient development of gaming mechanisms, including character movement, combat interactions, and AI behaviours. The simplicity of GDScript enables swift prototyping, while its performance optimisation guarantees smooth gameplay experiences. The advanced capabilities of GDScript, including class extension and reusable component creation, provide it as an essential tool for developing modular and scalable games.

1.4.3 Visual Design Tools: Aseprite

Aseprite facilitates the creation of pixel art and sprite animations, enabling the creation of detailed and aesthetically pleasing game components. The tool's emphasis on pixel art renders it an exceptional option for crafting characters, settings, and effects that align with the game's look. The intuitive interface and animation capabilities enable the efficient production of excellent visual elements.

1.4.4 Project Management Tools: Trello

Trello functions as a core project management tool, facilitating task monitoring, milestone scheduling, and workflow optimization. It endorses the Agile technique implemented for this project, guaranteeing adaptability and ongoing assessment of progress. By categorising tasks and tracking their completion, Trello helps maintain clear communication and focus throughout the project lifecycle.

1.4.5 Technologies That Were Considered

Several technologies were explored and analysed early in the project to determine the best tools for attaining the project's objectives. While the Godot engine and GDScript were eventually chosen, other engines and programming languages were examined but rejected for various reasons.

Unity and Unreal Engine were top contenders due to their robust feature sets and active development communities. Unity's versatility and C# scripting make it a market favourite, although Unreal Engine's graphics powers and Blueprint visual scripting system are unrivalled. However, both engines have a higher learning curve than Godot, especially for a single developer working on a time-sensitive project. Furthermore, Godot's lightweight design and emphasis on 2D/3D hybrid development better suited the scope and technical needs of this project.

1.5 Timeline (Milestones)

The timeline for the development of this game spans over the academic year, split into two terms. The project will be split into several phases to ensure that both the design and the implementation of the game are progressing well. Each phase will include specific milestones, ensuring that development stays on track and allows for flexibility for any unforeseen changes. As seen below:

1. **Pre-Production Phase (week 1-3)**

- **Objective: Establish a clear foundation for the project, such as conceptualising the game mechanics, setting up the development environment, and preparing the initial assets. Conduction of research will still be going on, as well as research into the game engine.**
    - **Week 1-2:**
        - **Initial project planning and setup.**
        - **Study Godot engine and mechanics along with papers relevant to the mechanics of the game.**
        - **Create the initial project design document.**
        - **Meet with the supervisor and finalise ideas.**
        - **Start the draft of the project plan.**
    - **Week 3:**
        - **Carry out learning on the Godot engine.**
        - **Complete tutorials of basic core functions.**
        - **Complete the project plan.**

2. **Core system development (week 4-10)**

- **Objective: Focus on developing the games core gameplay mechanics, including the menu screens for the player to navigate, player controls, enemies (and their AI), dungeon map creation, basic combat mechanics, and a character class and stats section. The goal here is to have a playable prototype by the end of this phase. Alongside all of this, it is important to make sure that the report is being written up while the development process is being carried out.**

- **Week 4-5:**
  - **Create the menus desired for the game, along with fluid connectivity between them.**
  - **Implement player movement and a basic combat system.**
  - **Test to see if these mechanics are suitably implemented.**
- **Week 6-7:**
  - **Set up the player HUD.**
  - **Begin development of enemies and their factories.**
- **Week 8-9:**
  - **Set up collision with player, enemies, and the placeholder map.**
  - **Test simple player, enemy, and map interaction.**
- **Week 9-10:**
  - **Develop the procedural generation of the dungeon.**
  - **Integrate player and enemy mechanics into procedural levels.**
- **Week 11-12:**
  - **Add any other features if there is time, such as power-ups, and start the development of the puzzles and mini games.**
  - **Refine some of the mechanics and refactor.**
  - **Consider all test cases for everything developed so far.**
  - **Prepare for the interim review, the 5000-word report and the demo code.**

3. Milestone: first playable prototype (end of term 1)

- **By the end of term one, there will be a working prototype, featuring the basic functions of the game such as exploration, combat, and rudimentary dungeon generation. In-house playtesting will begin at this stage, and the feedback from this will be collected to guide the development into the next term.**

4. Advanced features and refinement (Week 11-17)

- **Objective: In this phase I will refine core systems, add more complex features, such as advanced enemy AI, further intricate level design, and a more fleshed-out combat design. This phase will also contain debugging and playtesting.**
  - **Week 11-13:**

- **Consider the feedback from the playtesting and adjust the timeline to fit in these comments.**
- **Enhance the enemy AI (pathfinding, group behaviours).**
- **Add more enemy types with potential unique abilities (bosses (optional)).**

- **Week 14-15:**
  - **Implement the power-up and pickups system (inventory system).**
  - **Refine the procedural generation (environment obstacles, traps).**
- **Week 16-17:**
  - **Test all the mechanics in the game so far.**
  - **Bug fixing and early optimisation is to be conducted.**

5. **Milestone: Alpha version (mid-term 2)**

- **At this stage, the game should be feature complete in terms of the core elements, with all the major mechanics functional. The focus of the project will now shift to finalisation, fixing bugs, polishing the game, and if there is extra time, any additional optional features.**

6. **Polish and final testing (weeks 17-23)**

- **Objective: Focus on the refinement of the game experience, including the visuals, user interface, and any other extra features. Extensive playtesting will be done to make sure the game is polished.**
  - **Week 17-19:**
    - **Finalise all the assets requires**
    - **Polish procedural map generation.**
  - **Week 20-22:**
    - **Conduct final playtesting.**
    - **Resolve any remaining bugs and make final optimisations.**
  - **Week 23:**
    - **Prepare the final submission.**
    - **Prepare for the final report and demo.**
    - **Publish game.**

7. Milestone: Final submission (End of term 2)

- **The game should now be complete, along with the final report and the demo for it. Any remaining work will be concluded.**

Chapter 2: **Background Theory**

Roguelikes have long been highly regarded for their focus on replayability, challenging gameplay, and emergent narratives. At the core of their appeal is procedural generation, a technique that dynamically creates content such as levels, enemy configurations, and loot for each game algorithmically, rather than manually. This synergy between the roguelike genre and procedural generation fosters endless variability, ensuring that no two playthroughs are identical. Players are encouraged to adapt to unpredictable environments and situations, which adds depth and excitement to the gameplay experience. Procedural generation not only reduces the manual labour required for level design (11) but also aligns seamlessly with roguelike principles of exploration, adaptability, and tactical decision-making. By creating a unique and dynamic world, these games maintain player engagement while opening opportunities for creative design and technological innovation.

The variety of game genres emphasises the unique qualities meant to satisfy different player experiences. For example, although action games include fast-paced fighting and reflex-driven difficulties, role-playing games concentrate on narrative richness and character advancement. Particularly noteworthy are Roguelikes' focus on strategic gameplay, repeatability, and uncertainty. Procedural generation—which guarantees that every playthrough stays different and interesting—is a trademark of the roguelike genre. Though Rogue gave its name to the roguelike genre, it was not the first game to use procedural generating and permadeath (14). Still, its impact confirmed these concepts as a mainstay of the genre and highlighted their continuing appeal in games.

Not limited to roguelikes, procedural generation has been effectively applied in many genres. Procedural approaches are shown by games such as Minecraft and Diablo (2). Procedural generation in Minecraft controls not only the terrain but also the location of buildings and treasures by use of artificial intelligence (10). These illustrations show how effectively procedural generating could improve replayability and envelop players in dynamic and large environments.

Modern gaming also depends much on artificial intelligence (AI), which shapes the behaviour of non-player characters (NPCs) and improves the whole player experience. From simple pathfinding and opponent movement to complex decision-making systems modelled on human conduct, artificial intelligence finds expression in games. In dungeon crawlers and roguelikes, AI is essential for creating engaging challenges, as enemies must adapt dynamically to player actions while adhering to the game’s design principles. Millington (8) highlights the importance of balancing intelligence with predictability, ensuring enemies are formidable yet not frustratingly overpowered. Techniques like finite state machines and A* pathfinding algorithms are commonly employed to manage enemy behaviour, facilitating adaptive responses to player movement, combat, and exploration. In procedural environments, AI must also adjust to the unpredictability of the game world, ensuring cohesive and responsive gameplay. By integrating these AI principles, games create an immersive experience where players feel as though they are competing against intelligent adversaries rather than pre-programmed scripts.

Good and poor games vary mostly in their capacity to enthral players and offer a coherent, pleasurable experience. Well-designed games captivate players with their simple mechanics, meaningful challenges, and balance of accessibility and complexity. Effective level design depends, Adams (1) underlines, on well-defined goals, logical development, and a feeling of accomplishment. On the other hand, badly crafted games irritate players with imbalanced difficulty, unclear objectives, or repetitive surroundings that fail to motivate involvement.

A critical component of good game design is pacing—offering moments of intensity balanced with opportunities for exploration and reflection. Adams further notes that well-crafted levels subtly guide players, keeping them orientated while encouraging exploration. Conversely, games that rely on repetitive tasks or lack variety risk losing the player's interest. Visual and audio feedback also

play a vital role in reinforcing player actions and immersing them in the game world. Games with inconsistent design elements or insufficient polish can break immersion, ultimately alienating players. By adhering to these principles, games can deliver an engaging and cohesive experience that resonates with their audience.

With a unique mix of procedural generation, emergent gameplay, and strategic complexity, roguelikes have carved themselves a clear spot in the gaming industry apart from other genres. Their emphasis on unpredictability and repeatability never fails to excite players with experiences that seem novel and interesting with every run. While other genres depend on handmade levels or linear advancement, roguelikes flourish on the challenge and discovery that accompany dynamic, always-shifting environments. This flexibility not only fits the changing expectations of players but also guarantees the relevance of the genre in a market that celebrates creativity. As games like Hades and The Binding of Isaac demonstrate, roguelikes are not just a niche—they are a testament to the enduring appeal of games that challenge and reward in equal measure.

Chapter 3: **Current Implementation**

The development of the top-down dungeon crawler game is proceeding steadily, with numerous fundamental components currently established. This section defines the main systems and mechanics established to date, emphasising their functionality and integration within the game. The project establishes an effective and scalable foundation for future development by iteratively constructing and testing these essential components.

3.1 Menu Navigation

The main menu system functions as the player's initial engagement with the game and is designed for easy navigation. It includes essential selections, including Start Game, Options, and Exit, with each button leading to its corresponding function.

```
extends Control


# Called when the node enters the scene tree for the first time.
func _ready():
	$StartButton.connect("pressed", 			Callable(self, 
"_on_StartButton_pressed"))
	$OptionsButton.connect("pressed", 			Callable(self, 
"_on_OptionsButton_pressed"))
	$ExitButton.connect("pressed", 			Callable(self, 
"_on_ExitButton_pressed"))


func _on_start_button_pressed():
	#Loads the game state screen
	get_tree().change_scene_to_file("res://MainGameEnvironment.tscn")


func _on_options_button_pressed():
		#Loads the options menu
	get_tree().change_scene_to_file("res://OptionsMenu.tscn")


func _on_exit_button_pressed():
	# Quit the game
	get_tree().quit()
```

Buttons are connected to their respective functions via Godot's connect() method in the _ready() function. Each button triggers a specific action: starting the game transitions to the main game scene, accessing options loads the options menu, and exiting the game calls get_tree().quit(). This approach ensures a simple and efficient navigation structure.

```
func _ready():
	$BackButton.connect("pressed", 			Callable(self, 
"_on_BackButton_pressed"))


func _on_BackButton_pressed():
```

```
	get_tree().change_scene_to_file("res://control.tscn")
```

The above code is the options menu script, which is not fully functional as of yet but utilises the same concept with the back button.

```
func _ready():
	# Connect button signals
	$VBoxContainer/RestartButton.pressed.connect(_on_restart_button_pre
ssed)
	$VBoxContainer/QuitButton.pressed.connect(_on_quit_button_pressed)

func _on_restart_button_pressed():
	# Reload the current scene to restart the game
	get_tree().reload_current_scene()

func _on_quit_button_pressed():
	get_tree().quit()
```

The game over screen also utilises the same idea here.

3.2 The Player

The player system is a lot more complex and encompasses core functionalities such as health management, movement, and combat. There is a constant flag called “is_dead” that runs throughout the whole of this code.

3.2.1 Health Management

```
var max_health = 100
var current_health = 100
var is_dead = false
@onready var animation_player = $Knight/AnimationPlayer

func take_damage(damage):
	if is_dead:
		return
	current_health -= damage
	if current_health <= 0:
		die()
	else:
		update_health_ui()

func die():
	if is_dead:
		return
	is_dead = true
	animation_player.play("Death_B")
	print("The player has died")

	var hud = get_node_or_null("/root/StaticBody3D/HUD")
	if hud:
		hud.visible = false
		show_game_over()

func update_health_ui() -> void:
```

```
	var hud = get_node_or_null("/root/StaticBody3D/HUD")
	if hud:
		hud.update_health(current_health, max_health)
```

The player's health system monitors damage and handles death situations. The take_damage() function decreases health following an attack on the player and verifies if health reaches zero. Consequently, the die() function is triggered, executing a death animation and identifies the player as dead. The update_health_ui() function guarantees that the health indicator on the HUD accurately represents the current status, providing clear feedback to the player.

3.2.2 Game Over Handling

```
func show_game_over():
	var game_over_scene = load("res://GameOver.tscn")
	var game_over_instance = game_over_scene.instantiate()
	get_tree().current_scene.add_child(game_over_instance)
```

When the player dies, the show_game_over() function is called and displays the relevant screen. The screen is instantiated and added to the scene tree, ensuring a smooth transition into a final state where the player has lost the game.

3.2.3 Combat System

```
var attack_range = 2.0
var attack_damage = 10
func _process(delta):
	if is_dead:
		return
	if Input.is_action_just_pressed("attack"):
		attack()
		animation_player.play("1H_Melee_Attack_Chop")

func attack():
	var enemies_in_range = get_enemies_in_range()

	for enemy in enemies_in_range:
		if enemy.has_method("take_damage"):
			enemy.take_damage(attack_damage)

func get_enemies_in_range() -> Array:
	var enemies_in_range = []
	var space_state = get_world_3d().direct_space_state

	# Create a shape for the query (a sphere for melee range detection)
	var shape = SphereShape3D.new()
	shape.radius = attack_range  # Set the radius of the sphere

	# Create the query parameters
	var query = PhysicsShapeQueryParameters3D.new()
	query.shape_rid = shape.get_rid()  # Use the RID of the shape
	query.transform = global_transform  # Use the player's global
transform to place the sphere
	query.collision_mask = 1

	var result = space_state.intersect_shape(query)

	for hit in result:
		if hit.collider != self and
hit.collider.has_method("take_damage"):
			enemies_in_range.append(hit.collider)
```

```
	return enemies_in_range
```

The combat system allows the player to attack the enemies within a defined range. The attack() function identifies enemies within the player's vicinity by calling the enemies_in_range() function, which is implemented as a sphere shape to detect other colliders near the player. An enemy detected within that, that has a take_damage() method, are subsequently damaged.

3.2.4 Movement System

```
var speed = 5.0
var gravity = -9.8
func _physics_process(delta):
	handle_movement(delta)

func handle_movement(delta):
	if is_dead:
		return
	velocity.x = 0
	velocity.z = 0


	var input_dir = Vector3.ZERO

	if Input.is_action_pressed("move_forward"):
		input_dir.z += 1
	if Input.is_action_pressed("move_back"):
		input_dir.z -= 1
	if Input.is_action_pressed("move_left"):
		input_dir.x += 1
	if Input.is_action_pressed("move_right"):
		input_dir.x -= 1

	if input_dir != Vector3.ZERO:
		input_dir = input_dir.normalized()

	velocity.x = input_dir.x * speed
	velocity.z = input_dir.z * speed

	move_and_slide()

	if is_dead:
		return
	elif velocity.length() > 0:
		animation_player.play("Walking_A")
	else:
		animation_player.play("Idle")
```

The movement of the player is realised by capturing the user's input. The handle_movement() function calculates direction based on the input, applies gravity (if needed), and moves the player using the move_and_slide() function. Linked with this are the animations that are based upon the player's current velocity, such as idle and walking.

3.2.5 Animation Handling

```
@onready var animation_player = $Knight/AnimationPlayer

func handle_movement(delta)
if is_dead:
		return
```

```
	elif velocity.length() > 0:
		animation_player.play("Walking_A")
	else:
		animation_player.play("Idle")
```

The AnimationPlayer node oversees visual responses to player actions. Animations for walking, attacking, idlling, and dying are executed according to the player's state or input. For instance, initiating an attack activates a chopping animation, whereas walking and idling alternate based on movement. This enhances immersion by delivering responsive and fluid images.

3.3 The Enemy

3.3.1 Behaviour Management

```
func _physics_process(delta: float) -> void:
	if not is_instance_valid(player) or player.is_dead:
		return
	move_towards_player(delta)
	if is_player_in_range():
		attack_player()
```

This function orchestrates the enemy's behaviour. It is constantly checking to see if the player is alive and valid and then calls the move_towards_player() to approach the player. If the player is in range of the attack, it invokes attack_player().

3.3.2 Movement

```
func move_towards_player(delta: float) -> void:
	var direction = (player.global_transform.origin - global_transform.origin).normalized()
	velocity.x = direction.x * speed
	velocity.z = direction.z * speed
	velocity.y = 0
	move_and_slide()
```

The move_towards_player() function calculates the direction to the player by normalising the vector between the enemy's position and the player's position. It then applies velocity based on the direction and moves the enemy using move_and_slide().

3.3.3 Detecting The Player

```
func is_player_in_range() -> bool:
	var distance_to_player = global_transform.origin.distance_to(player.global_transform.origin)
	return distance_to_player <= attack_range
```

The function here calculates the distance between the enemy and the player using the distance_to() function. If the distance is less than or equal to the set attack_range, it returns true, letting the enemy know the player is within range.

3.3.4 Attacking The Player

```
func attack_player() -> void:
	if can_attack:
		player.take_damage(attack_damage)
```

```
            can_attack = false
            await get_tree().create_timer(attack_cooldown).timeout
            can_attack = true
```

The attack_player() function verifies whether the enemy is prepared to attack (can_attack). When triggered, it inflicts damage on the player by using the take_damage() function on the player object. Following the attack, the function assigns false to can_attack and utilises a timer to implement the cooldown period, after which the enemy is permitted to attack again.

3.3.5 Taking Damage and Death

```
func take_damage(damage):
      health -= damage
      if health <= 0:
            die()
```

This function reduces the enemy's health when the player attacks. Once the health reaches zero, the die() function is called, as seen below, to remove the enemy from the game state.

```
func die():
      queue_free()
```

3.3.6 Enemy Spawner

```
@export var enemy_scene: PackedScene
@export var spawn_interval: float = 3.0
@export var max_enemies: int = 5

var enemy_count = 0

func _ready():
      spawn_enemies()

func spawn_enemies() -> void:
      while enemy_count < max_enemies:
            var enemy_instance = enemy_scene.instantiate()

            var spawn_position = Vector3(
                  randf_range(-10, 10),  # X position
                  1,  # Y position (ground level)
                  randf_range(-10, 10)   # Z position
            )
            enemy_instance.global_position = spawn_position


            add_child(enemy_instance)
            enemy_count += 1
            await get_tree().create_timer(spawn_interval).timeout
```

The enemy spawner functions as a node that intermittently generates enemies at random locations within a designated radius. The spawn_enemies() function utilises the exported variables enemy_scene, spawn_interval, and max_enemies to regulate the spawning behaviour. It creates a new enemy from the PackedScene, designates a random spawn location within a specified range, and incorporates it into the scene. The spawner adheres to the max_enemies' restriction, guaranteeing that no more than the permitted number of enemies are active at once. Employing await with create_timer() adds a delay between each spawn, ensuring a consistent influx of adversaries during gameplay.

3.4 The HUD

```
extends CanvasLayer

func update_health(current_health: int, max_health: int) -> void:
      $TextureProgressBar.value = current_health
      $Label.text = "Health: %d / %d" % [current_health, max_health]
```

The HUD is implemented as a CanvasLayer, so it remains displayed on the screen regardless of the player's position in the game environment. Every time the player's health changes, the update_health() function dynamically updates both the health bar and a textual label. The TextureProgressBar visually depicts the current health, and its value attribute is set to the current_health parameter. Furthermore, the Label node displays the health as a numerical value in the type of Health: X / Y, where X represents the current health, and Y represents the highest health.

Chapter 4: **Software Engineering**

This project's software engineering methodology focuses on scalability, maintainability, and robustness in game design and implementation. This is especially essential because this is my first time utilising the Godot engine, a powerful yet unique tool with a node-based architecture and the GDScript programming language. This necessitated extensive consideration and exploration to determine its capabilities, limits, and best practices for attaining the project's objectives.

Throughout the development process, I emphasised modular design principles to ensure that separate components, such as combat, procedural generation, and AI systems, could be designed, tested, and integrated smoothly. Godot's open-source nature and copious documentation provided essential learning resources; nonetheless, integrating complex technologies such as procedural content creation and dynamic AI necessitated adapting old methods to this new context.

4.1 Software Methodologies

For this project, I used an Agile approach to oversee the development process. Agile was chosen because of its iterative and flexible nature, which complements the dynamic requirements of game production. By splitting the project down into small chunks, I was able to focus on individual aspects like enemy behaviour and combat mechanics while still adding regular testing and feedback. This strategy enabled ongoing refinement and adaptability, ensuring that the game progressed successfully over time.

Other techniques, such as the Waterfall model, were examined but found unsuitable for this project. Waterfall's linear form would have made it difficult to handle changes or unexpected obstacles that are typical in creative undertakings such as game development. Similarly, while Extreme Programming (XP) places a strong emphasis on testing and pair programming, its stringent principles and requirement for frequent cooperation would have been difficult to accomplish in a solo development setting.

Agile gave me the perfect blend of structure and flexibility, allowing me to respond to difficulties, enhance features, and prioritise jobs quickly. This iterative strategy ensured that every stage of development—from prototyping and testing to integration—was in line with the project's objectives and timeframes.

In addition to Agile, I applied Object-Orientated Programming (OOP) principles throughout the development process. OOP offered a solid foundation for developing modular and reusable code, which is critical for a project of this size. For example, enemy behaviour and player mechanics were implemented as separate classes to facilitate maintenance and scalability. This method supported Agile by keeping the codebase organised and flexible to changes.

4.2 Diagrams

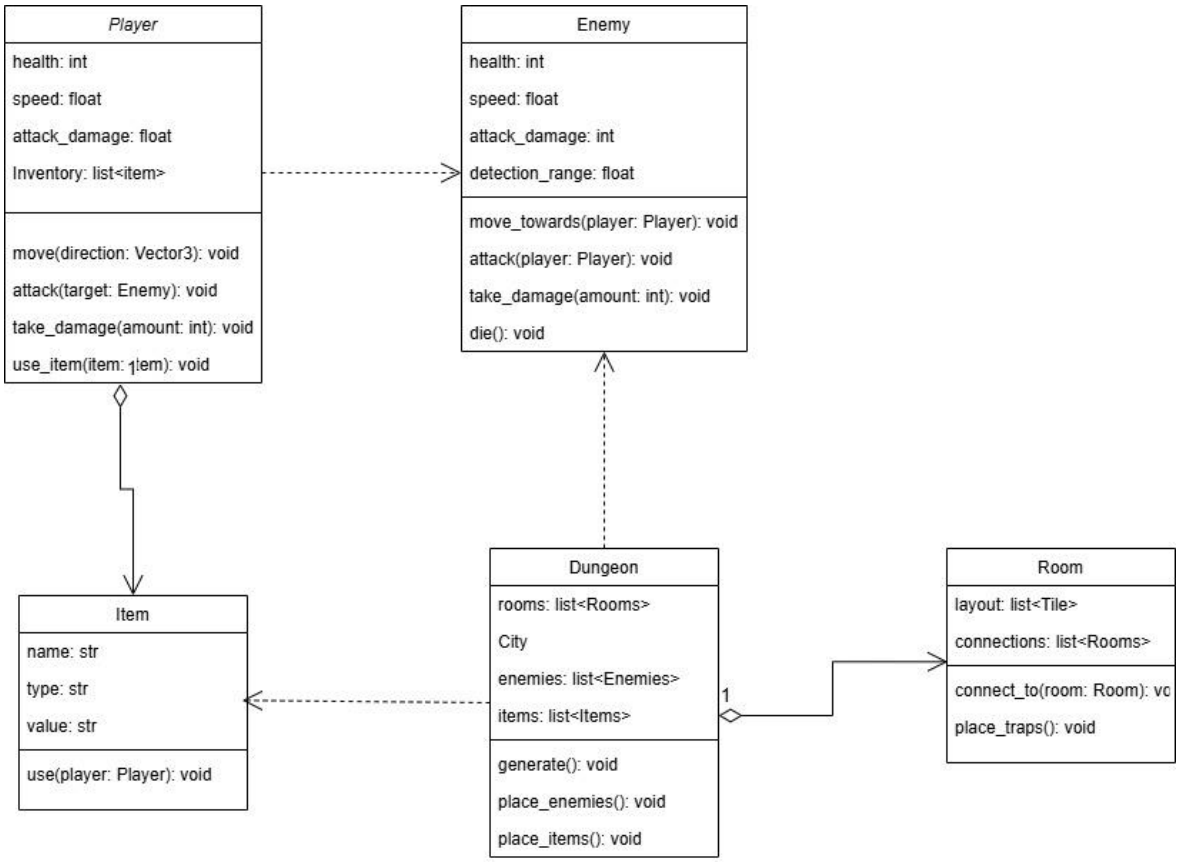

Figure 1: Class Diagram - Relationships between core game components

4.3 Testing

Testing has been an important part of the development process, ensuring that each feature works as expected and fits seamlessly into the game. A multi-tiered testing technique was implemented, including unit testing, integration testing, and playtesting, along with future testing plans such as feasibility testing, which allowed for a thorough review and refining of the game's mechanics and systems.

The battle mechanism, opponent behaviour, and procedural generation methods were all tested individually. These tests guaranteed that each module carried out its intended role in isolation, discovering mistakes early in the development process. To prevent unforeseen behaviours, the enemy's assault range, movement patterns, and damage logic were all extensively examined. Similarly, the procedural generation process, when implemented, will be tested with a wide range of settings to verify that each dungeon layout was both cohesive and passable.

Integration testing was performed to ensure that multiple game systems operated together. For example, the interactions between the player, opponents, and the health system were thoroughly tested to ensure accurate health reduction, proper game-over triggers, and smooth transitions between fighting and non-combat stages. This process was critical for discovering problems caused by dependencies or unanticipated interactions across modules, such as collision bugs or inaccurate health updates during combat.

Playtesting entailed manually playing the game to evaluate its user experience, gameplay balance, and overall flow. This step was extremely useful for detecting subtle issues, like pacing issues,

imprecise visual feedback, or unexpected difficulty spikes, that automated testing would have missed. Future playtesting will also give qualitative feedback on areas for development, for example, things such as improving combat mechanics, changing enemy AI behaviours, and adjusting the complexity of procedurally created levels. These sessions emphasised ways to improve the game's accessibility and overall enjoyment.

Feasibility Testing was planned to determine the game's performance and compatibility with various machine setups. Load times, memory utilisation, and frame rate stability are all important elements in providing a seamless and comfortable playing experience. Testing on computers with varied hardware capabilities will be carried out to uncover optimisation opportunities and keep the game accessible to a wider audience. For example, profiling techniques built into the Godot engine will be utilised to identify performance bottlenecks such as unoptimised assets or wasteful code routes. The procedural generation mechanism will receive special attention to guarantee that it runs smoothly and does not cause delays during level transitions. These tests will provide valuable insights into how the game works under various settings, guiding any necessary improvements.

Additionally, edge-case testing will be carried out to determine how the game handles uncommon conditions like quick inputs, overlapping opponent attacks, or excessively huge randomly generated landscapes. This ensures that the game is resilient and reliable under a variety of scenarios.

By iteratively employing these testing methodologies, the project maintains a high level of functionality and polish, resulting in a seamless and engaging player experience. The testing method will be ongoing throughout development to meet new challenges as features are introduced or enhanced. This iterative testing approach, combined with player input and feasibility testing, guarantees that the end result is both dependable and engaging.

4.4 Code Quality

Throughout the development process, excellent code quality has been prioritised to ensure the project's robustness, maintainability, and scalability. Several tactics and best practices have been used to reach this goal.

4.4.1 Modular Design

Code is divided into small, self-contained modules that each handle a single aspect of the game, such as player movement, combat, AI behaviour, or procedural creation. This modularity facilitates debugging and updating, as changes in one module have little impact on others. For example, the battle mechanisms and health systems are contained in reusable routines, allowing them to be customised for both player and opponent characters.

4.4.2 Coding Standards

Consistent naming conventions, unambiguous comments, and structured code formatting have been implemented to improve readability and collaboration. Variables and functions are named descriptively, such as take_damage() or get_enemies_in_range(), so the code's purpose is readily apparent. Comments are added to help clarify difficult logic.

4.4.3 Error Handling

Error management is carefully considered to avoid crashes or unexpected behaviour during gameplay. Functions contain checks for invalid states, such as ensuring that the player or enemy instance is legitimate before acting. For example, is_instance_valid() is used to check that objects exist before accessing their attributes or methods.

4.4.4 Testing and Iteration

The project includes unit and integration testing as part of the development process. To detect and address errors as soon as possible, both automated and manual tests are run on a regular basis. This iterative method ensures that new features are implemented while maintaining old functionality.

4.5 Version Control

Version control is an important part of the project since it ensures an organised and reversible development process. GitLab was chosen as the project's primary version control platform. GitLab's comprehensive capabilities, like branching, issue tracking, and continuous integration, make it a great solution for efficiently managing codebases.

Commit management is essential for tracking changes and recording progress. Descriptive and relevant commit messages guarantee that every change is fully documented, providing a chronological record of the development process. This method is especially valuable when returning to a project after a long absence, such as working on other university courses, because the thorough commit messages serve as a road map for understanding what was implemented, amended, or fixed.

Furthermore, GitLab's version control features make reverting changes simple. If a problem or unexpected behaviour is encountered, Git's ability to rollback commits or reset the codebase to a previous stable state allows for quick recovery. This has been quite useful during testing and feature implementation, as experimental modifications might occasionally impair functioning. Reverting to a known good state allows faults to be handled without jeopardising overall progress.

Chapter 5: **Reflections**

5.1 Reflection on Project Plan

Reflecting on the initial project plan, it has given a beneficial framework for guiding progress while allowing for flexibility in response to problems and new insights. The planned roadmap guaranteed that core components like combat mechanics and AI behaviour were prioritised and implemented within the timeframe specified. Certain components, such as procedural generation, which is a major feature of the game, have yet to be implemented. This delay happened because more time was needed to refine basic systems such as combat and enemy AI, assuring the project's stability before introducing more complicated features. In addition, there were additional university commitments and deadlines that required consideration.

The project's iterative nature also had an impact on its timeline. Frequent testing cycles revealed chances to tweak features, such as improving enemy behaviour and balancing battle mechanisms, which took longer than anticipated. While these iterations were not expressly accounted for, they proved critical to improving the game's quality.

Unexpected obstacles, such as debugging complicated interactions and optimising performance, necessitated additional revisions to the timeframe. Despite these obstacles, the plan's modular architecture allowed other areas of development to advance while addressing these issues. Moving forward, procedural generation and advanced features will remain the significant focus.

This thought emphasises the necessity of adaptability and the utility of a well-structured plan as both a guide and a tool for efficient problem solving.

5.2 Milestones Achieved

Several important milestones were successfully met during the development process:

- Core Combat System: The player's combat mechanics, such as attacking, damage calculation, and health management, have been incorporated and improved.
- Basic AI behaviour: Enemies can detect, approach, and attack the player. Basic patrol and combat techniques have been integrated.
- User Interface (UI): A functional HUD has been designed to display player health and provide feedback while playing.
- Menu Navigation: A fully operable main menu has been built to provide gamers with straightforward navigation.
- Code Structure: Key systems have been built with modular and reusable code to ensure scalability and maintainability in future updates.

These milestones provide a solid basis for the game, enabling the incorporation of more advanced features in subsequent phases of development.

5.3 Next Steps

The following phase of development will focus on finishing fundamental functionality and improving the overall playing experience. Key priorities include:

- Implementing Procedural Generation: Creating and integrating methods to generate dynamic dungeon layouts while assuring replayability and variation.

- Enemy AI enhancements include the introduction of more complicated behaviours such as ranged attacks, ambush tactics, and enemy coordination.

- Adding Puzzles and Traps: Creating intriguing puzzles and cleverly placed traps to challenge the player and enhance gameplay.

- Polishing Visuals and Audio: Improving animations, effects, and sound design to offer a more immersive experience.

- Extensive Testing and Optimisation: Conducting rigorous testing to ensure seamless performance across various hardware configurations, as well as fixing any outstanding bugs or performance concerns.

By tackling these elements in a systematic manner, the project will proceed towards its ultimate aim of creating a polished, entertaining, and replayable dungeon crawler game.

Chapter 6: **Bibliography**

[1] Adams, E., & Rollings, A. (2010). *Fundamentals of game design*. New Riders.

[2] Blizzard Entertainment. (1996). *Diablo* (Version 1.0) [Video game]. PC. Blizzard Entertainment.

[3] Blizzard Entertainment. (2000). *Diablo II* (Version 1.0) [Video game]. PC. Blizzard Entertainment.

[4] Dahlskog, S., Björk, S., & Togelius, J. (2015). Patterns, dungeons and generators. In *Foundations of Digital Games Conference, FDG, Pacific Grove, USA (2015)*. Foundations of Digital Games.

[5] Godot Engine Documentation. (2024). *Godot Engine Documentation*. Retrieved from https://docs.godotengine.org/.

[6] Izgi, E. (n.d.). *MASTER THESIS Framework for Roguelike Video Games Development*. [online] Available at: https://dspace.cuni.cz/bitstream/handle/20.500.11956/94724/120289816.pdf?sequence=1&isAllowed=y.

[7] McMillen, E. (2011). *The Binding of Isaac* (Version 1.666) [Video game]. PC. Edmund McMillen.

[8] Millington, I. (2019). *AI for Games*. CRC Press.

[9] Nystrom, R. (2014). *Game programming patterns*. Genever Benning.

[10] Patterson, B., & Ward, M. (2022). Adaptive Procedural Generation in Minecraft.

[11] Shaker, N., Togelius, J., & Nelson, M. J. (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer International Publishing.

[12] Smith, G. (2014, April). Understanding procedural content generation: a design-centric analysis of the role of PCG in games. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

[13] Valtchanov, V., & Brown, J. A. (2012, June). Evolving dungeon crawler levels with relative placement. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*.

[14] Wikipedia contributors. (2024, November 25). Rogue (video game). In *Wikipedia, The Free Encyclopedia*. Retrieved 19:17, December 12, 2024, from https://en.wikipedia.org/w/index.php?title=Rogue_(video_game)&oldid=1259508873

Chapter 7: **User Manual**

Prerequisites:

Before running the game, make sure to have the Godot 4 engine installed: download from: here

Setting up the project:

USING GIT:

1. Open terminal/command prompt.
2. Clone the repository.
3. Navigate to the project folder.

WITHOUT GIT:

1. Download the project as a .zip file from the repository.
2. Extract the contents to a folder on your system.

Running the game in Godot:

1. Open the Godot engine.
2. Click import project and find the folder containing the relevant files.
3. Select the file and open
4. Once opened, press F5 to run the game

Chapter 8: **Appendix**

8.1 Project Diary

30/09/2024

The initial ideas for this game are that it is to be built in a game engine such as Unreal or Godot. Research into both of these have been done but more thorough work is to be done. Ideas for the game are a FPS, drawing inspiration from titles such as Doom or Half-life. Other ideas do include top-down dungeon crawlers with puzzle-based elements.

02/10/2024

With much consideration to the ideas and functionality of both engines I have decided on using Godot. This engine is more suitable for me and this project and aligns with the skills I possess to excel with some of the concepts. The documentation for this engine has been read and tutorials provided have been attempted. Further with this the project plan will be started, finding relevant books and papers to aid in my work.

04/10/2024

On the 3rd of October I had met with my supervisor and discussed the project ideas that I had come up with and talked about any relevant help with the project plan. Following this a draft plan has been started.

07/10/2024

The plan timeline has been drafted up and the abstract has been partially drafted too. Research into relevant papers has also been conducted with papers such as 'Designing procedurally generated levels' - Linden, R., Lopes, R., & Bidarra, R. (2021). Designing Procedurally Generated Levels. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 9(3), 41-47. https://doi.org/10.1609/aiide.v9i3.12592. Risks and mitigations for those risks have been drafted into the daybook.

08/10/2024

The risks and mitigations section of the project plan has been completed. Once other parts of the plan are complete a draft will be sent to my supervisor for any final comments and tweaks and will be changed. with this, the timeline will now be started, building upon the initial draft I did for it. Abstract is still to be finished as the bibliography for it is still limited.

09/10/2024

The timeline for the project has been completed, with specified milestone and mapped to the term weeks. The abstract still needs work and further research is being conducted. Once a rough abstract is complete, I will send it to my supervisor for comments, and with these, on the 10th the plan will be completed, ready for submission on the 11th.

I have sent what I have done of the plan so far to my supervisor for further guidance, once feedback is received from them, I will continue in the correct direction.

10/10/2024

I have received positive feedback about my plan from my supervisor with further tips to finish off the abstract and bibliography. Work on this is underway and should be completed by tonight. Further research on papers have also been done.

The project plan is pretty much finished. The timeline, risks and bibliography is done, the abstract is almost finished, it needs a touch up and a bit more writing and the plan will be ready for the submission. Once done, the IDE will be setup and coding will begin.

11/10/2024

The project plan has been completed and uploaded to Moodle. Now that it is complete, the coding for the game will begin.

The file has also been uploaded to the Gitlab documents directory.

The interim report will be drafted once the coding starts.

13/10/2024

The Godot IDE is setup, linked with the GitLab, a master branch has been created for the initial work to begin. I will start as stated in the project plan, with the game's menu screens. I have created the buttons with a control node and have attached a script to allow for functionality of these menu buttons, this is not finished yet.

14/10/2024

I have created placeholder menu screens such as the main menu, game screen, and the options menu. I have partially scripted the menu buttons to be connected to each screen when they are pressed.

I am facing an error while getting the buttons to correspond to their appropriate screen, the change_Screen functions.

I have realised the documentation I was using was for Godot v3 and not v4 which is the version I am using. in v3 they used just change_scene, whereas in v4 they changed it to change_scene_to_file("..."), which now works. I will now refer to using the v4 documentation, opposed to the v3 which I wrongly opened. The buttons for the main menu screen now have functionality and take the me to the correct screen. I will now begin on creating a back button for the menus and visually sort them out.

I have centred the main menu buttons and gave them labels.

I have added a back button to options menu, created a script in the options menu to add functionality to get the player back to the main menu.

15/10/2024

Today I looked at some potential free assets that I can use for the UI, however still undecided on which to go for. I have created a 3D player with a mesh and collision nodes, I will now add functionality to these. I also attempt to do the player movement and try to attempt the above style camera.

I have created a basic rudimentary movement script for the player and mapped the physical buttons on the keyboard to work with the script. I will work on the camera and test to see if it works.

I have created a test environment, basic plane with collision, the movement for the player works

along with collision, the camera however is not working as expected. Therefore, I need to work on fixing that.

I have sorted the camera angle out but now the controls seem to be inverted; I will need to look into this and fix it.

I have had to do a temporary inversion of the controls which has seem to do the job, I have sorted out the camera angle as well. I have replaced the placeholder game menu with the current screen with the player and environment.

21/10/2024

After a few days off to work on other university work and commitments, it is the start of week, and I will start the next stages of the work required. This week I am to implement the player HUD, implement the enemies and a basic combat system.

22/10/2024

I have created a 2D scene for the players HUD, with a placeholder health bar. The rest of the HUD will be added soon. I have also linked and overlayed it on the main game screen, having tested it, it is functional. I cannot test this health bar till I create some enemies, so my next goal is to create a simple enemy factory.

23/10/2024

I have worked on the enemy 3D model and given it a script, which lets it move randomly and have speed and health. I then created a spawn factory in the main game state which will spawn the enemy randomly on the map. This is all tested and works as needs be. One error encountered was the enemy kept spawning into the map then falling below, this was a simple fix by changing its y coords higher.

I had some time to work on the assets, I added textures to the buttons and gave them a slight animation of hovered and pressed. I also added some basic lighting to the main game state along with some colour differentiations.

I had a look at some further assets such as animations for the player. I'm still yet to decide on one but I may get a free animated one and just use it as a placeholder and use it to understand how to implement the animations in the project.

24/10/2024

I implemented a very basic combat system where the player presses the space bar near an enemy and they take damage, once zero they are killed and taken off the screen. I will continue to improve and work on this combat system, but this very basic system will suffice for now.

28/10/2024

To start off the new week I spent some time reflecting on what had been done and what needs doing, consulting the timeline I created in the project plan. I have realised that functionality wise I am ahead on what I had predicted which is a good sign, which means I can spend some time on finding and implementing assets and animations into the game.

I have visited websites such as Kenney and itch.io for free assets to use.

28/10/2024

I have started to work on the enemy attack script.

12/11/2024

Coming back to the project I have finished the enemy attack's function and their pathfinding. Fixed errors that arise such as the players attack mechanism affecting the players health instead of the enemies. Also removed redundant comments left for myself when starting off as they no longer serve any purpose.

The next steps to follow is to work on the procedural generation of the dungeon itself and finalise and apply the assets.

18/11/2024

I have been spending my time researching procedural map generation these past few days and watching YouTube videos on it. I'm gaining a better understanding of how to implement it and the usage of it.

28/11/2024

I have been making separate projects to test the procedural generation. so far, it's very simple and not exactly how I want it and figuring out how to implement it into my code and project is difficult.

04/12/2024

I have found some assets and animations to use as placeholder for now for my game. I have imported them and tested to see if they are compatible, which they are.

I have added simple animation and the model to the player, which are functional and work, only thing that isn't is the attack animation. Once this is done, I will continue work on my separate file on procedural map generation.

11/12/2024

Today I had my project demonstration for the interim submission.

12/12/2024

The interim report is all written and the creation of the video demo will be done as well. All ready for the submission date on the 13th.

8.2 Link to Demo Video

https://youtu.be/MvgKArEBm2k