# Tribhuvan University

# Academia International College

**Final Year Project Report**

**On**

**Desktop Game (reDrift: The Lost Dhaka)**

**[CSC 412]**

**Under the supervision of**

**“Mr. Bishwas Mathema”**


**Submitted by**

**Bigyan Prasad Adhikari (T.U. Exam Roll No. 26484/077)**

**Sohail Raj Maharjan (T.U. Exam Roll No. 26519/077)**


**Submitted to**

**Department of Computer Science and Information Technology**

**Academia International College**

**Institute of Science and Technology**

**Tribhuvan University**

**Janauary, 2025**

# Tribhuvan University

# Academia International College


**Final Year Project Report**

**On**

Desktop Game (reDrift: The Lost Dhaka)

[CSC 412]

A final year project submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University

Submitted by

Bigyan Prasad Adhikari (T.U. Exam Roll No. 26484/077)

Sohail Raj Maharjan (T.U. Exam Roll No. 26519/077)

Submitted to

Department of Computer Science and Information Technology

Academia International College

Institute of Science and Technology

Tribhuvan University

**Janauary, 2025**

# Tribhuvan University

**Institute of Science and Technology**

# Academia International College

**Department of Computer Science and Information Technology**

Email: mail@academiacollege.edu.np


## Supervisor's Recommendation

I hereby recommend that the project work report prepared under my supervision by Mr. Bigyan Prasad Adhikari (26484/077) and Mr. Sohail Raj Maharjan (26519/077) entitled "reDrift: The Lost Dhaka" be accepted as fulfilling in partial requirements for the degree of Bachelors of Science in Computer Science and Information Technology. In my best knowledge, this is an original work in Computer Science and Information Technology.

……..

Mr. Bishwas Mathema

Project Supervisor

HOD/Program Coordinator

Department of Computer Science and Information Technology

Academia International College

Gwarko, Lalitpur

# Tribhuvan University

## Department of Computer Science and Information Technology

# Academia International College

## Certificate of Approval

This is to certify that this project prepared by Mr. Bigyan Prasad Adhikari and Mr. Sohail Raj Maharjan entitled "reDrift: The Lost Dhaka" in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

| | |
|---|---|
| ……..<br>Mr. Bishwas Mathema<br>Project Supervisor<br>Department of Computer Science and IT<br>Academia International College | ……..<br>Mr. Bishwas Mathema<br>HOD/Program Coordinator<br>Department of Computer Science and IT<br>Academia International College |
| ……..<br>Internal Examiner<br>Academia International College | ……..<br>External Examiner<br>Central Department of CSIT<br>Tribhuvan University |

# Acknowledgement

We owe my most profound appreciation to Academia International College for giving us a chance to work on this project as part of our syllabus.

Special thanks to our supervisor, Mr. Bishwas Mathema (HOD/Program Coordinator, Academia International College), for his consistent guidance, support, and feedback throughout the report's creation. We are generously obligated to him for providing this excellent opportunity to expand our knowledge. It helped us a lot to realize what we studied for.

We would like to express our sincere gratitude to all those individuals, families, friends, colleagues, and teachers for supporting and helping us a lot in finalizing this project within the limited time frame by providing valuable insights and feedback on the report.

Thanking You,

Bigyan Pd. Adhikari (T.U. Exam Roll No. 26484/077)
Sohail Raj Maharjan (T.U. Exam Roll No. 26519/077)

# Abstract

ReDrift: The Lost Dhaka is a new top-down RPG game that was created in the desktop collaborative process in order to fully plunge players into an immersive and inspired experience. It follows a protagonist from Nepal on a journey to regain the fragments of Dhaka, which is a treasured cultural artifact now broken and scattered across different realms. Through this journey, the player goes through the game by being availed with skillfully designed maps and meeting challenges such as puzzles, combat, dungeon exploration, etc., with a final battle at the end to get back the last piece, thus restoring Dhaka. It was created on modern game development platforms by Godot for engine design, Aseprite for sprites' creation, and Krita for asset designing. Key features in core gameplay included a strong combat system, pathfinding AI for enemy interaction, and a health and healing system that dynamically changed. The game narrative is interweaved with cultural aspects to better enhance the gaming experience and thus forge a much deeper connection with the story. reDrift: The Lost Dhaka is a perfect example of how one 2-person team blended technical expertise, creative storytelling, and teamwork to come up with a great project. With iterative design and comprehensive testing, it showcases the potential that lies in culturally inspired game development in the industry.



***Keywords: top-down RPG, sprites, Godot Engine, Krita, Aseprite, pathfinding, AI.***

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| DLC | Downloadable Content |
| GDD | Game Design Document |
| GDScript | Godot Script |
| NPC | Non Playable Character |
| UI | User Interface |

# Chapter 1: Introduction

## 1.1. Introduction

reDrift: The Lost Dhaka is a two dimensional top down role playing game developed using the free and open source game making engine “Godot”. The Godot game engine makes use of its own self built-in game scripting language named GDScript which aids the developers in making their game more functional and lively.

The RPG genre of the game means that it provides the storytelling while also allowing the rich experience to the player about how an RPG should be. The game focuses on the small intricate levels designed connected together along with other aspects of an RPG such as diverse environments, clean combat system with entities, interaction with other entities such as NPCs. This genre based developed game blends the old-school vibes of generic RPG games produced over the last few decades. The manually build world and different scenes and levels are crafted using different tools in compliment to Godot, such as Aseprite and Krita. By using the node based architecture of the Godot engine, the development of the game was made easy to create seamless and smooth transitions, collisions and interactions with the dynamic environment in the game.

## 1.2. Problem Statement

The old school methods of game development of similar genres required very powerful knowledge of programming as well as logic building. With the development of this project, it aims to show how the Godot engine can be used to simplify this complex and hectic process while providing output of high quality at the same time. Some key challenges include:

- Providing a game without pay-to-win model.
- Implementing efficient character movement and collision detection
- Creating a narrative based progressing game story
- Developing an engaging combat system
- Managing game state and save/load functionality

## 1.3. Objectives

The main objectives of reDrift: The Lost Dhaka game include:

- Create a fun RPG game with a good story, proper gameplay, and a system that helps players grow in the game.
- Make sure all players can enjoy the full game without paying extra money.
- Add creative gameplay, fun challenges, and a unique story to make the game different from others.

## 1.4. Scope and Limitations

### 1.4.1. Scopes

The scopes of reDrift include:

- Player character movement and animation
- Basic combat system with melee and ranged attacks
- Inventory system with equipment slots
- NPC interaction system
- Quest tracking system
- Save/load functionality

### 1.4.2. Limitations

The limitations of reDrift include:

- Single-player gameplay only
- Limited to 2D graphics
- Fixed camera perspective

## 1.5. Methodology

The Agile methodology has been used during the development of this project, as it focuses more on iterative and flexible process of development, also dividing the projects into smaller pieces of steps known as sprints. Each sprint was carried out to focus on quick development of working, manageable and functioning part of the game system. The project was also carried out during development by communicating closely with the supervisor by gaining

insightful advices and feedback regularly. The main focus maintained during the project was finalizing the scope and requirements of the game development project as well as figuring out how long it would take to complete the entire project.

Initially, the planning phase comprised of writing down the project objectives and the requirements. Then, strategic planning was done on how to achieve the plans set out to be carries during the project. Then concurrently development of each components of the game system was done along with proper testing.

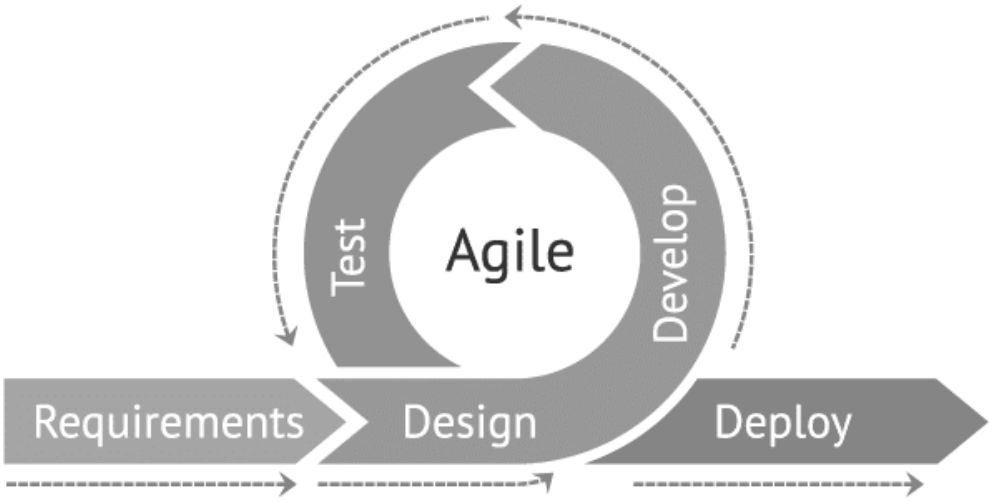


**Figure 1.1: Agile Methodology of Software Development**

## 1.6. Report Organization

After successfully completing the project, a comprehensive project report has been prepared. The report begins with essential introductory sections, including the Title Page, Certificate Page, Acknowledgement, Abstract, Table of Contents, and lists of Abbreviations, Figures, and Tables.

The main body of the report is structured into six chapters, each focusing on a specific aspect of the project:

Chapter 1: Introduction

This chapter provides an overview of the project, covering key aspects such as the project introduction, problem statement, objectives, scope and limitations, and methodology.

Chapter 2: Background Study and Literature Review

Here, the project's background is discussed, along with a review of existing literature. This

includes summaries of relevant projects, research papers, and articles that helped shape the project's direction.

Chapter 3: System Analysis

This section delves into system analysis, including requirements and feasibility studies. It defines the system's functional requirements using a use case diagram and presents a Gantt chart to visually illustrate the timeline and progress of various project tasks.

Chapter 4: System Design

This chapter explores the design phase in detail, covering the implementation process, model architecture, user interface, and system interactions. It also includes insights into the algorithms used in the project.

Chapter 5: Implementation and Testing

The focus here is on the implementation process and testing phases. It provides an overview of the tools and dependencies used to build the system and outlines the testing procedures undertaken to ensure functionality.

Chapter 6: Conclusion and Recommendations

The final chapter wraps up the project with a summary of key findings and conclusions. It also highlights potential areas for future improvements and enhancements.

The report concludes with a References section, formatted in IEEE style, and Appendices containing system screenshots and important source code snippets.

# Chapter 2: Background Study and Literature Review

## 2.1. Background Study

Recent trends pointing to pay-to-win models in the gaming industry have also gotten especially worrying of late. This has indeed significantly shaped the gaming experience and is proving to be a source of constant irritation for many players in an increasingly unbelievable manner. Most of the models seem to tilt more in favor of money-making for the players, thereby leaving an alarming gap between those who can afford to pay and those who cannot afford it. It brings a sort of imbalance in the playing field, very much times the feeling that skill and effort are shadowed by financial investment.

Moreover, Frustration is further added when a compelling story lacks its essence and a game design fails to ignite the experience that it promises. Most of these games in 2D, which were once the mainstay of the industry, seem to have very little going for them today. The modern landscape of games really seems to have been overtaken by three-dimensional games. It automatically gives the feeling that 2D games are either old-fashioned or simply do not seem to be improving. However, it is not a problem of the 2D format by itself—it is the failure on the part of developers to innovate in this space. Most 2D games today do not have the depth, immersive storytelling, and polished mechanics that are necessary for competing with their 3D counterparts. Many features, like exclusive skins, powerful boosts, and unlocked content, all sit behind paywalls—turning gaming into an expensive burden rather than a fun hobby.
Also, the concentration of a developer upon revenue generation has surmounted the very basic purpose of these games: for entertainment, immersion, and joy. The technique eventually becomes oblivious to the gameplay mechanics, innovative design, and experience given by the players if it is only driven by economics. Lack of quality storytelling, lack of visually appealing art styles, or lack of engaging gameplay is enough to keep players from being loyal to a game and to keep the potential newcomers away repeatedly.

This lack of interest in crafting enjoyable games has been the cause of the loss of countless loyal gaming communities and the stagnation of growth in fresh audiences. The modern way of development in video games really does cry out for change: it needs to be reborn with a focus on players and their experiences. This is about creating games that are strong in storytelling, have good gameplay mechanics, are creatively designed, and accessible at no

cost. It should especially touch base with 2D games that need to re-establish their relevance through exploration, combat mechanics, and player experience at par with 3D counterparts. A more considerable change could not only rekindle 2D gaming but also raise the bar for the entire gaming industry. There is a pressing need to recreate games at their roots: adventures, fun, and connection for people irrespective of their backgrounds.

## 2.2. Literature Review

Vampire Survivors has brought a fresh twist to the 2D action RPG genre with its innovative automated combat system and unique progression mechanics. The game stands out for its clever use of procedural content generation and character advancement, all while delivering an addictive and engaging gameplay loop. Its success has set a new standard for creating RPG mechanics that are both accessible and rich in depth, making it an excellent example of what can be achieved using the Godot Engine. [1]

Stardew Valley is a landmark achievement in indie game development, seamlessly blending RPG elements with life simulation mechanics. Its success lies in its well-executed systems, including inventory management, character relationships, and time-based events, offering valuable lessons for modern 2D RPG development. The game showcases how a solo developer can leverage modern tools to build complex and polished game systems. Its farming and crafting mechanics are especially notable for their ability to deliver interconnected gameplay that keeps players engaged while optimizing the use of computational resources. [2]

CrossCode is a great example of how to add complex real-time combat to a 2D RPG. The game mixes puzzles and combat in a way that works well and runs smoothly. It uses physics-based puzzles and combat, showing how to use Godot's physics engine in smart ways. The developers shared their process through blogs and articles, giving helpful tips for solving common problems in 2D RPGs, like making combat feel good and designing smart enemies. [3]

Ocean's Heart is a great example of how to create quest-based stories and use the environment to tell a story in a 2D top-down game. The way it was built, using modular design and smart resource management, is very helpful for indie developers working with modern game engines. The game shows how to add features like dynamic weather, day-night cycles, and interactions with the environment, all of which can be done using Godot. Its way of designing the world and managing quests gives useful ideas for making open-world games that are fun to explore and run efficiently. [4]

# Chapter 3: System Analysis

## 3.1. System Analysis

Before starting the development of the game, a comprehensive task list was created, considering both functional and non-functional aspects. This was followed by an analysis of how the intended system (game) should operate to ensure an optimal player experience.

### 3.1.1. Requirement Analysis

The requirements can be functional and non-functional.

i. Functional Requirements

The project considered the following functional requirements:

- Interface for menus
- Ability to transition through scenes
- Progression and quest tracking
- Combat mechanics and battle system
- Interaction with NPCs and objects in the game world

ii. Non-Functional Requirements

The project considered the following non-functional requirements:

- The game should be accessible and playable offline, with the option to sync progress online.
- The game should have an intuitive and user-friendly interface for seamless navigation.
- The game should run smoothly on low to mid-spec systems, ensuring accessibility for a wide audience.
- The game should feature optimized performance for minimal load times and lag.

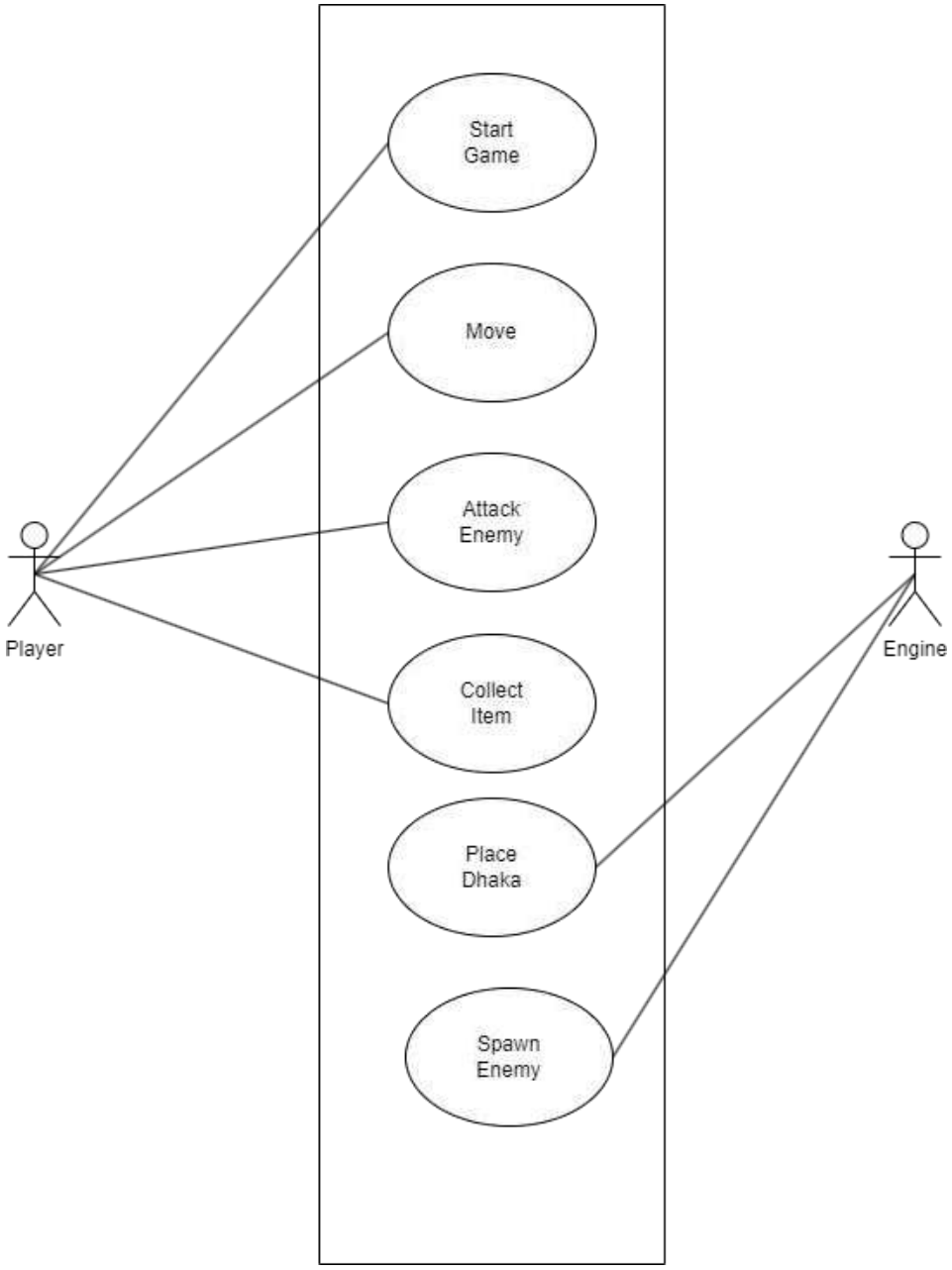


**Figure 3.1: Use Case Diagram**

The use case diagram indicates interaction between the Player and the game system. The Player may start the game, move, attack enemies, and collect items. The engine spawns collectibles and enemies and ensures that the game environment reacts dynamically to the actions of the player. Each oval represents a particular use case or function within the game.

### 3.1.2. Feasibility Analysis

Feasibility analysis is conducted to determine whether a project is technically, economically, operationally, and schedule-wise feasible to guarantee successful project completion.

#### i. Technical Feasibility

The project is technically feasible, as it aligns with the latest technologies, both hardware and software. The game is developed using the Godot Engine with GDScript, a powerful open-source framework for 2D game development. The project utilizes modern graphic design tools like Aseprite and Krita for creating the game's assets, ensuring high-quality visuals. The system requirements are well within reach, ensuring compatibility with a wide range of personal computers. The game also integrates seamless interaction between user inputs and game logic, utilizing Godot's built-in physics system and node-based architecture for optimized performance.

#### ii. Operational Feasibility

Operationally, Godot Engine, with its straightforward interface and easy-to-learn scripting language, facilitates the development and operation of the game, reducing the time-to-market. The development team, consisting of a game developer and a designer, worked collaboratively to ensure smooth progress throughout the project. The game is designed to be compatible across different PC platforms, ensuring broad accessibility. The intuitive game mechanics and user interface ensure that players of all skill levels can easily navigate and enjoy the game. The integration of multiple difficulty levels and adaptive challenges enhances user engagement, promoting long-term playability.

#### iii. Economic Feasibility

The project is economically feasible due to the use of free and open-source tools, such as the Godot Engine for game development and Aseprite for sprite creation. These resources minimize development costs while maintaining high-quality production values. The game's system requirements are modest, ensuring it can run efficiently on low to mid-tier PCs.

#### iv. Schedule Feasibility

The project was completed within the estimated timeframe, accounting for all necessary tasks including game design, asset creation, programming, and playtesting. With a well-

planned timeline, the project was divided into manageable milestones, ensuring progress was made efficiently and within the given schedule.

The following Gantt chart depicts the schedule established for the development of the system.

**Table 1.1: Activity Schedule**

| Task | Start Date | End Date | Duration |
|---|---|---|---|
| Project Planning | 9/15/2024 | 9/28/2024 | 13 |
| Asset Creation | 9/29/2024 | 10/12/2024 | 13 |
| Story and Map Design | 10/13/2024 | 10/19/2024 | 6 |
| Core Mechanic Development | 10/20/2024 | 11/02/2024 | 13 |
| Character Design and Development | 11/03/2024 | 11/09/2024 | 6 |
| Pathfinding Integration | 11/10/2024 | 11/16/2024 | 6 |
| Cpmbat System Development | 11/17/2024 | 11/30/2024 | 13 |
| Playtesting Phase | 12/01/2024 | 12/14/2024 | 13 |
| Final Polishing | 12/15/2024 | 12/21/2024 | 6 |
| Beta Testing | 12/22/2024 | 12/28/2024 | 6 |
| Final Adjustments | 12/29/2024 | 01/04/2025 | 6 |
| Final Release | 01/05/2025 | 01/11/2025 | 6 |
| Report | 9/15/2024 | 1/13/2025 | 120 |

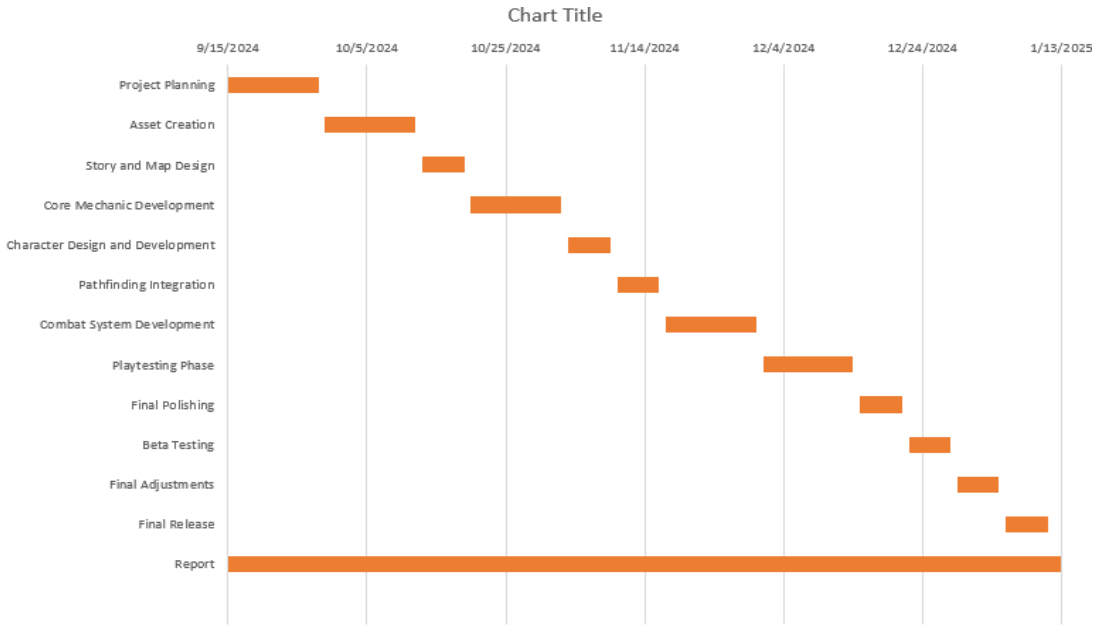


**Figure 3.2: Gantt Chart**

### 3.1.3. Object Modeling using Class and Object Diagrams

This game development project follows an object oriented approach because of the language being used to develop the game, GD Script is an object oriented programming language.

The following class and object diagram depicts the different classes used within the game component and the scene tree which allows the game to function smoothly. The player is connected to all other classes and interacts with collectible, enemy as well as the UI manager.

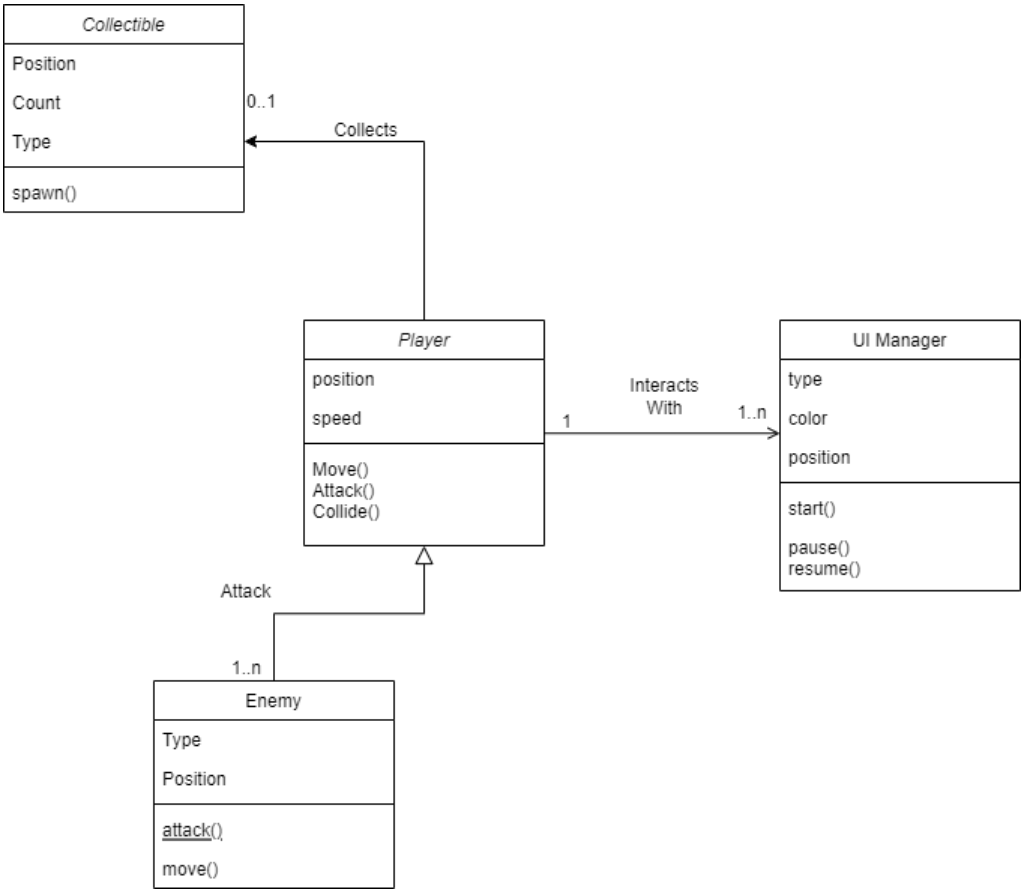


**Figure 3.3: Class Diagram**

The UI Manager provides a user interface to the player for attacking multiple enemies and collecting their collectibles that spawn within the game. The Enemy behaviors entail attacking and moving, and the Collectible has attributes such as position, count, and type.

### 3.1.4. Dynamic modelling using state and sequence diagram

The below state diagram and sequence diagram graphically depict the flow of processes used to capture, manipulate, store, and distribute state of game between the system and among its components.

The state diagram clearly depicts how the different state of the game are interconnected and how one another affects the game state whereas the sequence diagram represents the sequence of flow of different actions occurring during the game and its inter connectivity between the elements.

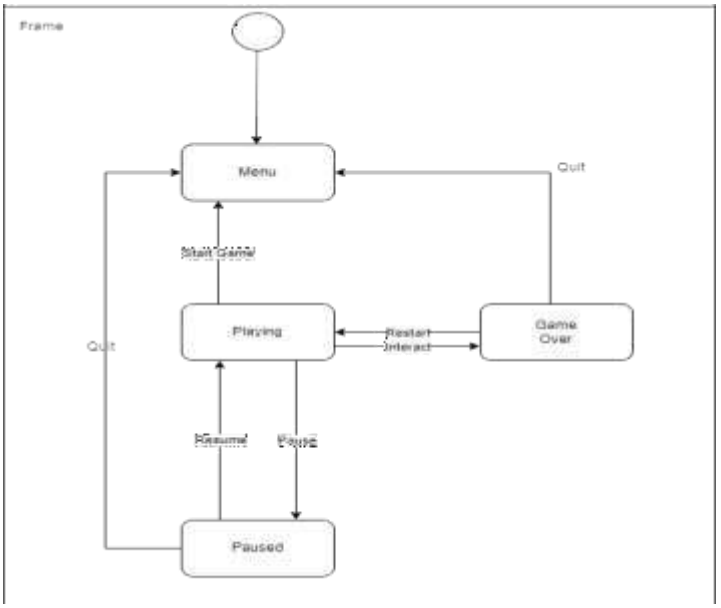


**Figure 3.4: State Diagram**

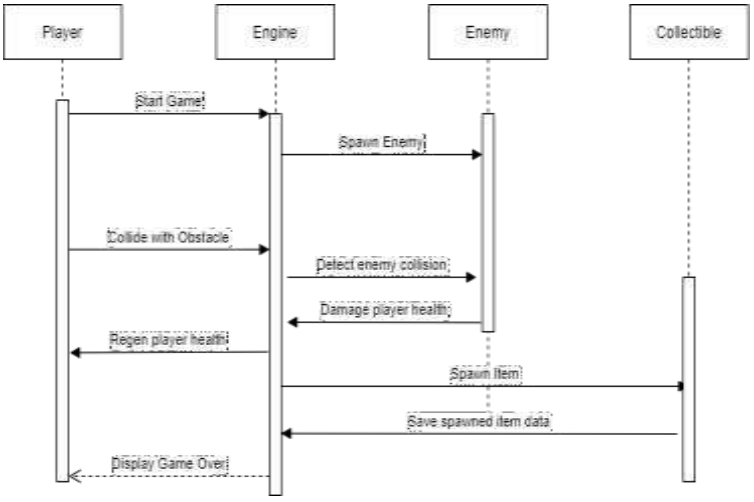


**Figure 3.5: Sequence Diagram**

### 3.1.5 Process modelling using activity diagram

Activity diagrams are simply an advanced from of flow-chart diagram that model the working of the system from one activity to another. The following activity diagram shows a graphical form of our game working. The flow of the process starts by initiating the game and controlling the player where the player can interact with other entities such as an enemy or a collectible.

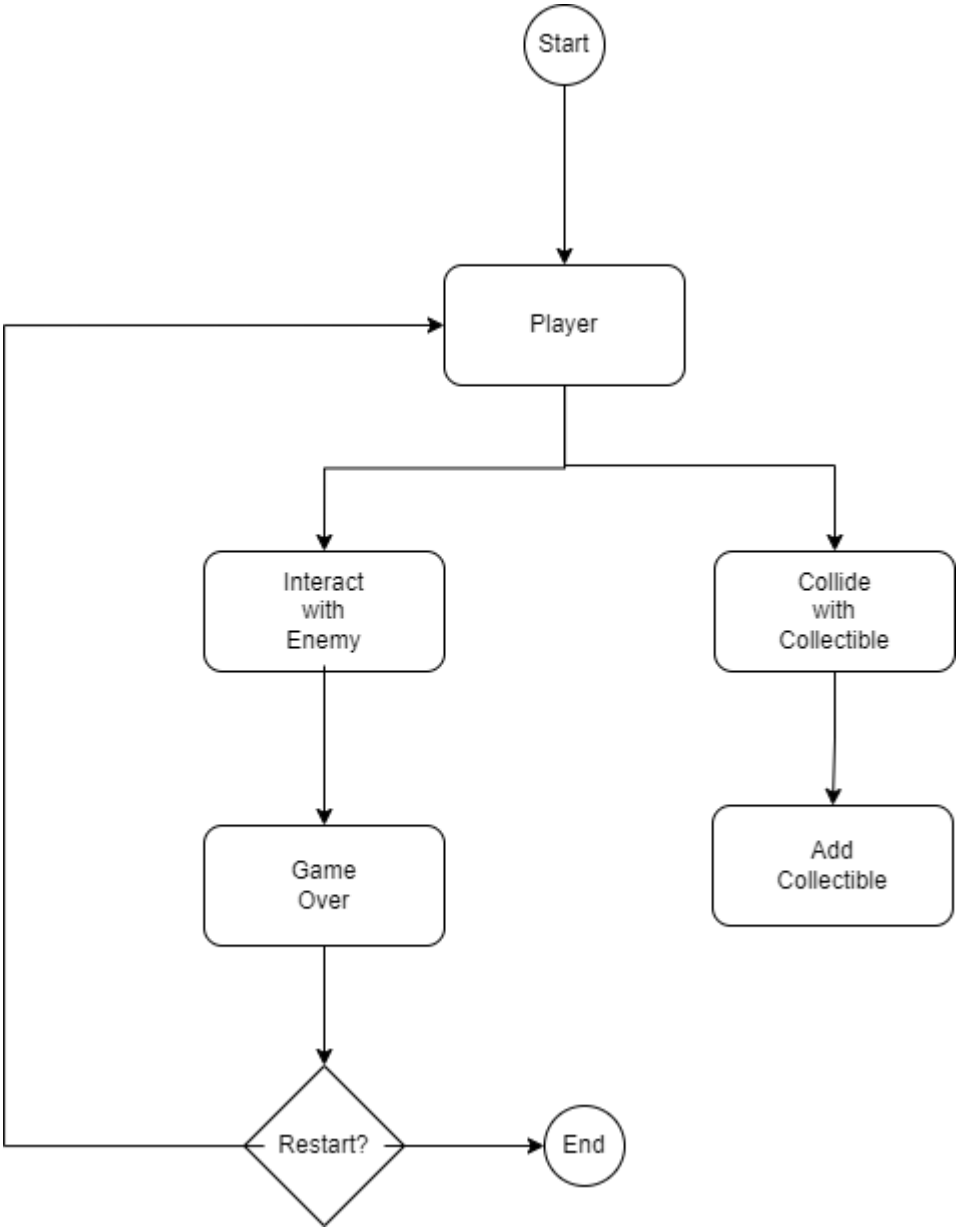


**Figure 3.6: Activity Diagram**

The activity diagram depicts a flowchart of the process starting from the game start to the ending of the game. In-between are the different events and logical situations that occur during the running phase of the game such as the player interacting with the enemy and the collectible.

# Chapter 4: System Design

## 4.1. Design

As discussed in the analysis chapter, the system design also follows an object-oriented approach.

- Refinement of Class, Object and Activity Diagram

  Since this is a very basic project with not much complexity, the class diagrams, object diagrams and activity diagrams designed earlier are already refined enough to be applicable within the project.

- Component Diagram

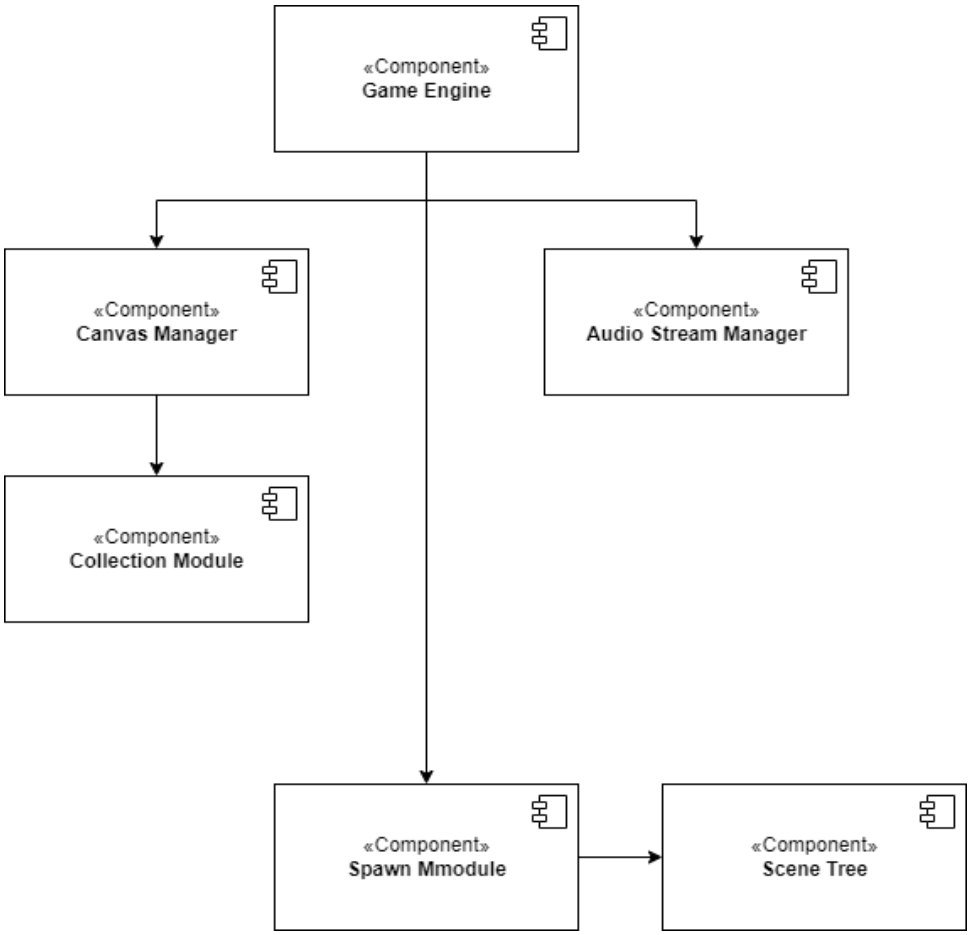


**Figure 4.1: Component Diagram**

The above component diagram of our project shows the diagrammatic representation of how the different components are interlinked together to function as a whole game. The game engine of Godot helps in rendering all the other components such as the canvas manager, collection module, audio stream manager as well as the scene tree.

## 4.2. Algorithm Details

### 4.2.1. Enemy AI Algorithm/Path Finding

The Enemy AI Algorithm is a system that controls the enemy characters within a game environment. It's responsible for finding paths around obstacles and leading enemies to their target. The pathfinding mechanism used is a grid-based A* algorithm, which calculates the best way to any one destination, considering both dynamic movement and obstacles.

It was meant to pose a bit more of a challenge and a bit more realistic behavior on the part of the AIs for the game. The workings of the algorithm are elucidated below:

1. Grid Representation**:** The game environment is divided into a grid of cells where each cell has a cost. These costs denote an obstacle or open space.
2. Starting Point: In the pathfinding process, the starting point is taken as the initial position of the enemy.
3. Goal: The current position of the player or any designated target within the environment is set as the goal.
4. Cost Calculation: The algorithm calculates two costs for every possible move:
    - g-cost: The expense of moving from the start point to the current cell.
    - h-cost: Estimated cost from the current cell to the target, provided by the heuristic function (e.g., Manhattan distance).
5. Path Selection: The algorithm picks a cell that has the lowest total cost: g-cost + h-cost. So, this becomes the path on which the enemy will move.
6. Obstacle Detection: When an obstacle is detected, the AI calculates a new path by rerouting around the obstacle.
7. Continuous Update: The pathfinding process is continuously updated while the player is moving so that the enemy can find the best way to the target.

### 4.2.2. Random Generation Algorithm

An algorithm of this nature would place different objects into the game dynamically—be these obstacles, enemies, or power-ups—randomly, across a specific area. The algorithm sets a standard in the variety of the experiences a player can expect in different gaming

sessions. The procedure is outlined below:

1. Define Boundaries: Establish the game's playable area and section it off to randomly place objects in said area.
2. A random value selects the object to spawn. This value shall lie in a certain range to spawn an obstacle or a collectible.
3. The cooldown mechanism applies a cooldown for every spawn of the object type.
4. The location is assigned by selecting random locations for every spawned object within those boundaries, eliminating clashes with other objects and obstacles.
5. History tracking makes sure the same object type is not repeated, providing diversity in the game environment.
6. So this process will continue uninterrupted throughout the course of the game and continue introducing new objects into the environment.

Both of these algorithms, when used in conjunction, will effectively make the world feel more dynamic, with smarter and more interesting behavior from the AI and elimination of predictability—thereby creating engaging and interesting experiences for the player.

# Chapter 5: Implementation and Testing

## 5.1. Implementation

2D RPG development was set up in a few stages that are quite common to all: formulation of basic gameplay mechanics, character systems, combat mechanics, inventory management, quest system, and UI implementation. These elements were then all developed in an iterative process with continuous testing and refinement using the built-in tools and debugging features of Godot Engine.

### 5.1.1. Tools Used

- Godot Engine 4.x: Primary game development engine
- GDScript: Primary programming language for game logic
- Aseprite: Pixel art and animation creation
- Git: Version control system
- GitHub: Repository hosting and collaboration
- Trello: Project management and task tracking
- Audacity: Sound effect editing and processing
- VS Code: Code editor with GDScript extensions

### 5.1.2. Implementation Details of Modules

Some of the modules included in the project are:

- Player Character Module: Handles player movement, animation states, and interaction with the game world. Implements character attributes, leveling system, and equipment management. The module uses Godot's KinematicBody2D for precise movement control and collision detection.
- Combat System Module: Manages real-time combat mechanics, including weapon handling, damage calculation, and enemy AI. Implements different attack patterns, hitbox detection, and combat animations using Godot's built-in animation system.
- Quest System Module: Manages quest tracking, progress updates, and reward distribution. Uses a signal-based system for quest event handling and updates. Implements save/load functionality for quest progress persistence.

- UI System Module: Handles all user interface elements including menus, inventory displays, and status indicators. Uses Godot's Control nodes for responsive UI layouts and theme customization.

## 5.2. Testing

Testing was carried out on parameterized game balances and had feedback and optimization based on the performance requirements and low operational bugs. There were multiple ways to go about testing the game to verify different aspects.

### 5.2.1. Test Cases for Unit Testing

In unit testing, individual testable units of code, typically at the function or method level, were tested to ensure their correctness and proper behavior.

The following test scenarios were employed for conducting unit testing.

**Table 5.1: Test Cases for Character Movement Testing**

| Test ID | Test Scenario | Expected Result | Observed Result | Test Status |
|---|---|---|---|---|
| 1 | Verify character moves in all directions | Character moves in all directions. | Same as expected result | Pass |
| 2 | Check animation state transitions | Character has animation while moving. | Same as expected result | Pass |

**Table 5.2: Test Cases for Combat System**

| Test ID | Test Scenario | Expected Result | Observed Result | Test Status |
|---|---|---|---|---|
| 1 | Validate enemy AI behavior | Enemy attacks and damages player | Player is not damaged | Fail |
| 2 | Validate enemy AI behavior | Enemy attacks and damages player | Same as expected result | Pass |

**Table 5.3: Test Cases for Enemy Pathfinding**

| Test ID | Test Scenario | Expected Result | Observed Result | Test Status |
|---|---|---|---|---|
| 1 | Enemy moves and locates the player position | Enemy reaches the player position | Enemy couldn't locate the player position | Fail |
| 2 | Enemy moves and locates the player position | Enemy reaches the player position | Same as expected result | Pass |

### 5.2.2. Test Cases for Integration Testing

After successful unit testing, integration testing was conducted to determine the collective functionality of each component unit as a unified project. The interaction between different components within the system were examined.

**Table 5.4: Test Cases for Integration Testing**

| Test ID | Test Scenario | Test Steps | Expected Result | Observed Result | Test Status |
|---|---|---|---|---|---|
| IT1 | UI and game state synchronization | Press play button and the main game scene loads | The main scene starts and user can play | Same as expected result | Pass |
| IT2 | Audio and visual system coordination | Game music plays when menu opened | The music starts playing in the main menu | Same as expected result | Pass |

### 5.2.3. Test Cases for System Testing

During system testing, a detailed assessment of the complete operational process of the system was carried out to detect errors and bugs, guaranteeing the proper functioning of the system as a whole. The entire system was developed and tested on a different machine than the development environment to guarantee satisfactory performance and efficiency. Hardware and software components were combined and tested as a whole.

The following test scenarios were employed for system testing.

**Table 5.5: Test Case for Performance Testing**

| Test ID | Test Scenario | Test Steps | Expected Result | Observed Result | Test Status |
|---|---|---|---|---|---|
| ST_PC | Load time measurement | Install and open the application and check the time duration | Successful implementation of the executable file | Same as expected result | Pass |

**Table 5.6: Test Cases for Playability**

| Test ID | Test Scenario | Expected Result | Observed Result | Test Status |
|---|---|---|---|---|
| ST_GB | Game balance evaluation | The game is well balanced with respect to the user skill | Same as expected result | Pass |
| ST_UI | User interface accessibility | Responsive when on clicked on icons and buttons, performing the desired action | Same as expected result | Pass |

## 5.3. Result Analysis

Testing results revealed the successful implementation of key game features across several important metrics:

- Performance Metrics: The game maintained a stable 60 FPS on target platforms, with memory usage kept within acceptable limits and load times under 3 seconds. Animation transitions were smooth and fluid.

- User Experience: Players experienced intuitive controls and responsive gameplay, complemented by clear UI feedback and engaging combat mechanics. The save/load functionality was stable and seamless.
- Technical Achievement: All planned features were successfully implemented, with efficient resource management, robust error handling, and a modular, maintainable codebase that ensures long-term game scalability.

These results confirm the game's readiness for deployment, with all core functionalities thoroughly tested and refined.

# Chapter 6: Conclusion and Future Recommendations

## 6.1. Conclusion

In essence, reDrift: The Lost Dhaka development is a successful project where creative storytelling mingled with engaging gameplay mechanics to provide players with a unique 2D RPG experience. The project has managed to achieve the goal of producing an engaging and immersive game, filled with dynamic challenges and other features such as combats, puzzles, and character progressions. The choice of tools for creating the game—Godot, Aseprite, and Krita—shows in the visually stunning assets of this game. Otherwise, the gameplay was smooth and consistent.

## 6.2. Future Recommendations

Some aspect of the game that can be improved and recommendations include:

- Expand the storyline with new realms, characters, and quests to increase replayability and deepen the narrative.
- Add multiplayer modes (co-op or competitive) to boost player engagement and expand the audience.
- Enhance AI with adaptive behaviors and complex pathfinding for a more challenging gameplay experience.
- Implement procedural level generation to offer varied and unique experiences with each playthrough.
- Port the game to additional platforms such as consoles and mobile devices to reach a wider player base.
- Develop downloadable content (DLC) with new areas, challenges, and storylines to sustain player interest and increase revenue.

Incorporating these recommendations will allow reDrift: The Lost Dhaka to grow into a larger franchise, securing its place in the gaming world and inspiring future projects.

# References


[1] Yong, O. (2022) Vampire survivors - development roadmap overview - steam news, Welcome to Steam. Available at: https://store.steampowered.com/news/app/1794680/view/3131696199131643254 (Accessed: 2 January 2025).

[2] King, N. (ed.) (2021) Stardew valley, Stardew Valley. Available at: https://www.stardewvalley.net/ (Accessed: 19 December 2024).

[3] By, S. (2022) CrossCode, Radical fish games. Available at: https://www.radicalfishgames.com/?p=2973 (Accessed: 19 December 2024).

[4] long, N. (2020) Ocean heart World generation, Nordcurrent. Available at: https://nordcurrent.com/games/oceans-heart/ (Accessed: 21 October 2024).

# Appendices

## Screenshots of the Application

### Main Menu


**Figure 4.1: Main Menu**

### How To?


**Figure 4.2: How To**

**Main Game Scene**

**Figure 4.3: Main Game Scene**

**Game Over Scene**


**Figure 4.4: Game Over Scene**

## Algorithm

### A* Algorithm

```
# Function to perform A* pathfinding algorithm
Function AStar(startNode, endNode, grid):
    # Create open and closed sets
    openSet = [startNode]
    closedSet = []

    # Initialize g, h, and f scores for each node
    startNode.g = 0
    startNode.h = calculateHeuristic(startNode, endNode)
    startNode.f = startNode.g + startNode.h

    # Loop through the openSet until it's empty
    while openSet:
        # Get the node with the lowest f score
        currentNode = getLowestFScoreNode(openSet)

        # If the current node is the endNode, we found the path
        if currentNode == endNode:
            return reconstructPath(currentNode)

        # Move currentNode from openSet to closedSet
        openSet.remove(currentNode)
        closedSet.append(currentNode)

        # Check the neighbors of currentNode
        for neighbor in getNeighbors(currentNode, grid):
            # If the neighbor is in the closedSet, skip it
            if neighbor in closedSet:
                continue

            # Calculate the tentative g score for the neighbor
```

```
            tentativeGScore = currentNode.g + calculateDistance(currentNode, neighbor)

            # If the neighbor is not in the openSet or the new g score is better
            if neighbor not in openSet or tentativeGScore < neighbor.g:
                # Update the neighbor's scores and set its parent
                neighbor.g = tentativeGScore
                neighbor.h = calculateHeuristic(neighbor, endNode)
                neighbor.f = neighbor.g + neighbor.h
                neighbor.parent = currentNode

                # Add the neighbor to openSet if it's not already there
                if neighbor not in openSet:
                    openSet.append(neighbor)

    # If there's no path found
    return []

# Function to calculate the heuristic (estimated distance to the end node)
Function calculateHeuristic(node, endNode):
    return abs(node.x - endNode.x) + abs(node.y - endNode.y)      # Using Manhattan
distance as heuristic

# Function to calculate the actual distance between two nodes
Function calculateDistance(node1, node2):
    return abs(node1.x - node2.x) + abs(node1.y - node2.y)

# Function to get the neighbors of a node
Function getNeighbors(node, grid):
    neighbors = []
    # Get the neighboring nodes (up, down, left, right)
    for dx in [-1, 1]:
        if grid.isWalkable(node.x + dx, node.y):
            neighbors.append(grid.getNode(node.x + dx, node.y))
    for dy in [-1, 1]:
```

```
            if grid.isWalkable(node.x, node.y + dy):
                neighbors.append(grid.getNode(node.x, node.y + dy))
        return neighbors

    # Function to get the node in the open set with the lowest f score
    Function getLowestFScoreNode(openSet):
        lowest = openSet[0]
        for node in openSet:
            if node.f < lowest.f:
                lowest = node
        return lowest

    # Function to reconstruct the path from the end node to the start node
    Function reconstructPath(node):
        path = []
        while node != null:
            path.insert(0, node)
            node = node.parent
        return path
```

**Random Spawn Algorithm**

```
# Function to spawn random entities in a grid-based environment
Function spawnRandomEntity(grid, entityPrefab, spawnCount):
    for i in range(spawnCount):
        # Randomly choose a position within the grid bounds
        randomX = rand_range(0, grid.width - 1)
        randomY = rand_range(0, grid.height - 1)

        # Check if the chosen position is walkable
        if grid.isWalkable(randomX, randomY):
            # Spawn the entity at the random position
            spawnEntity(entityPrefab, randomX, randomY)
```

```
# Function to spawn the entity at a specific position
Function spawnEntity(entityPrefab, x, y):
    entity = entityPrefab.instance()
    entity.position = Vector2(x, y)
    getParent().add_child(entity)
```

**Code implementation of A* using GDScript**

```
# A* Pathfinding Algorithm in GDScript

# Function to perform A* pathfinding algorithm
func AStar(start_node, end_node, grid):
    # Create open and closed sets
    var open_set = [start_node]
    var closed_set = []

    # Initialize g, h, and f scores for each node
    start_node.g = 0
    start_node.h = calculate_heuristic(start_node, end_node)
    start_node.f = start_node.g + start_node.h

    # Loop through the openSet until it's empty
    while open_set.size() > 0:
        # Get the node with the lowest f score
        var current_node = get_lowest_f_score_node(open_set)

        # If the current node is the end_node, we found the path
        if current_node == end_node:
            return reconstruct_path(current_node)

        # Move current_node from open_set to closed_set
        open_set.erase(current_node)
        closed_set.append(current_node)

        # Check the neighbors of current_node
```

```
        for neighbor in get_neighbors(current_node, grid):
            # If the neighbor is in the closed_set, skip it
            if neighbor in closed_set:
                continue

            # Calculate the tentative g score for the neighbor
            var tentative_g_score = current_node.g + calculate_distance(current_node, neighbor)

            # If the neighbor is not in the open_set or the new g score is better
            if not open_set.has(neighbor) or tentative_g_score < neighbor.g:
                # Update the neighbor's scores and set its parent
                neighbor.g = tentative_g_score
                neighbor.h = calculate_heuristic(neighbor, end_node)
                neighbor.f = neighbor.g + neighbor.h
                neighbor.parent = current_node

                # Add the neighbor to open_set if it's not already there
                if not open_set.has(neighbor):
                    open_set.append(neighbor)

    # If there's no path found
    return []

# Function to calculate the heuristic (estimated distance to the end node)
func calculate_heuristic(node, end_node):
    return abs(node.position.x - end_node.position.x) + abs(node.position.y - end_node.position.y)  # Using Manhattan distance as heuristic

# Function to calculate the actual distance between two nodes
func calculate_distance(node1, node2):
    return abs(node1.position.x - node2.position.x) + abs(node1.position.y - node2.position.y)

# Function to get the neighbors of a node
func get_neighbors(node, grid):
```

```
    var neighbors = []
    # Get the neighboring nodes (up, down, left, right)
    for dx in [-1, 1]:
        if grid.is_walkable(node.position.x + dx, node.position.y):
            neighbors.append(grid.get_node(node.position.x + dx, node.position.y))
    for dy in [-1, 1]:
        if grid.is_walkable(node.position.x, node.position.y + dy):
            neighbors.append(grid.get_node(node.position.x, node.position.y + dy))
    return neighbors

# Function to get the node in the open set with the lowest f score
func get_lowest_f_score_node(open_set):
    var lowest = open_set[0]
    for node in open_set:
        if node.f < lowest.f:
            lowest = node
    return lowest

# Function to reconstruct the path from the end node to the start node
func reconstruct_path(node):
    var path = []
    while node != null:
        path.insert(0, node)
        node = node.parent
    return path
```

**Random Entity Spawn Algorithm in GDScript**

```
# Random Entity Spawn Algorithm in GDScript

# Function to spawn random entities in a grid-based environment
func spawn_random_entity(grid, entity_prefab, spawn_count):
    for i in range(spawn_count):
        # Randomly choose a position within the grid bounds
```

```
        var random_x = rand_range(0, grid.width - 1)
        var random_y = rand_range(0, grid.height - 1)

        # Check if the chosen position is walkable
        if grid.is_walkable(random_x, random_y):
            # Spawn the entity at the random position
            spawn_entity(entity_prefab, random_x, random_y)

# Function to spawn the entity at a specific position
func spawn_entity(entity_prefab, x, y):
    var entity = entity_prefab.instance()
    entity.position = Vector2(x, y)
    get_parent().add_child(entity)
```