UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Author

Damian Skrzypek

Supervisor

Dr. James Patten

Faculty of Science and Engineering
Department of Computer Science and Information Systems
University of Limerick

Submitted to the University of Limerick for the degree of
B.Sc. Computer Games Development

2021

Abstract

Title: Shacetha: 2D Game Development

Shacetha is an investigation into what features make a game engaging, enjoyable and interesting to players, how to implement these features during the different stages of video game development including designing, prototyping and play testing and finally how the design and game changes during the implementation.

A game was created using the Unity game engine through an iterative process to showcase findings from this investigation.

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Irish or foreign examination board. All brands, product names, or trademarks are properties of their respective owners.

Damian Skrzypek
14 April 2021

Acknowledgements

I would like to thank my supervisor Dr. James Patten for letting me take on a game development project for my final year project. His support and advice throughout the whole project have been very valuable and is greatly appreciated.

I would also like to thank my parents for their support throughout the final year project and through all my time at the university.

Lastly, I want to thank all my friends for their support and the feedback they provided throughout this project.

Table of Contents

Table of Tables

Table of Figures

1 INTRODUCTION

1.1 GAMEPLAY INTRODUCTION

Shacetha is a 2D top-down arena game, with mechanics from different genres of games, the main influences come from the Hack and Slash and Rogue-lite genres, but it also contains some limited Role-Playing Games (RPG) features. Some examples of games in these genres include:

- UnderMine (https://store.steampowered.com/app/656350/UnderMine/).
- Moonlighter (https://store.steampowered.com/app/606150/Moonlighter/).
- Hades (https://store.steampowered.com/app/1145360/Hades/).

1.1.1 Top-Down Arena

The game is played by using a keyboard and mouse, and the player sees the game world from a top-down perspective i.e. the camera looks down on 2D sprites that represent the different objects in the world e.g. the player and enemies.

All of the combat takes place in constrained areas called arenas, an example of such arena can be seen in Figure 1. The main mode in the game is a an endless arena mode in which the player will fight multiple waves of enemies, with some tougher enemies called 'bosses' appearing every few waves and the arena will change as the player defeats more waves.

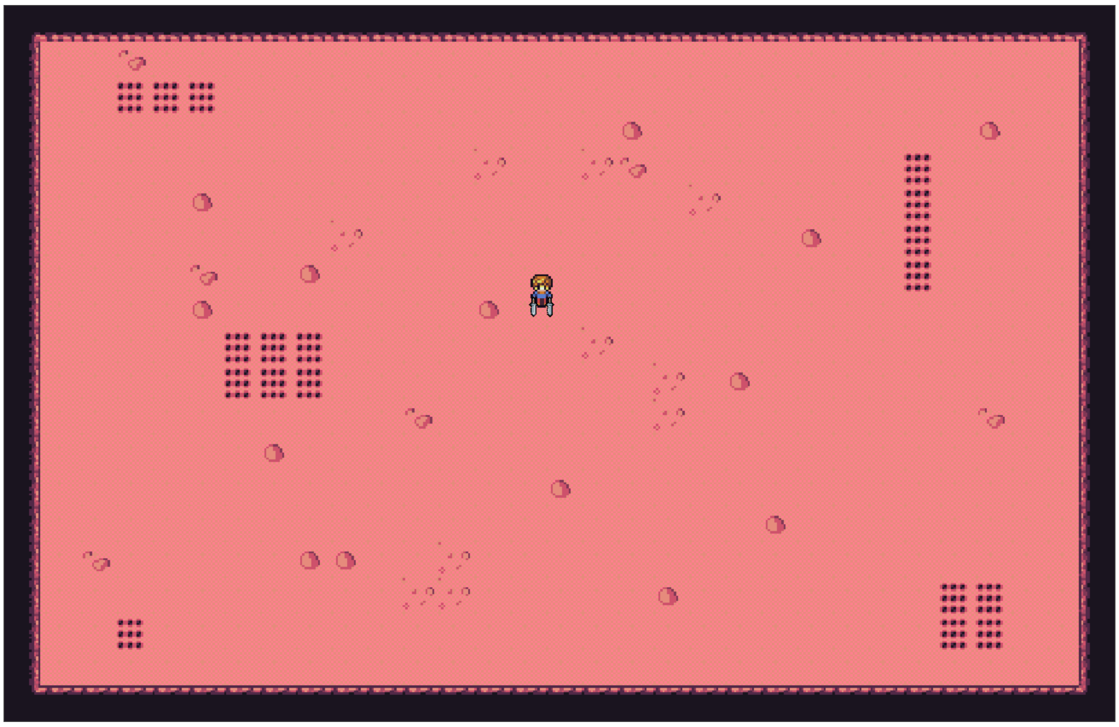


Figure 1. Example of an Arena

1.1.2 Hack and Slash Mechanics

Hack and Slash games are characterised by their emphasis on melee combat, with limited projectile weapons or attacks and an overall focus on combat over story elements with many different enemy types. Another common characteristic of Hack and Slash games is that they often give players a choice of different playstyles based on weapons.

Shacetha is focused on the combat allowing players to acquire many items of different types, that they can equip in either hand to fit to their playstyle, this can include equipping long range weapons like spears to keep enemies at a distance or using shorter sword to deal more damage in a wider area. The story is also limited to act as a short tutorial for the player.

1.1.3 Rogue-Lite Mechanics

Rogue-lite is a subgenre of games that take elements from rogue-likes, for Shacetha the main mechanic that is present from that subgenre is permadeath in the arena mode, this means that when a player dies they have to start over from arena/level 1.

Permadeath can feel punishing and un-rewarding but many rogue-lite games use different ways to make the process of restarting engaging, there are two main ways Shacetha tries to solve this problem i.e. game keeps track of the highest arena reached by the player, motivating the player to start another arena game to try to beat their previous score or to beat a score their friends might of achieved (Wang and Sun 2012), the game also includes many different items and enemies, as a player plays more they learn the enemy patterns allowing them to be more prepared for their attacks and achieve higher levels, this learning allows the player to feel that they are progressing in the game by improving their own skill level (Koster 2013).

1.1.4 RPG Mechanics

Shacetha includes some limited RPG elements, this includes item and inventory system as well as stats to provide the player with a progression as they defeat more waves of enemies, these mechanics also allow enemies to change as the game progresses as they are also capable of changing the items they are holding without changing their main behaviour.

1.2 Motivations

Designing and creating a video game from start to finish is a very interesting and challenging process, it covers many different areas of computer science depending on the type of game, what platforms it targets, the desired audience, and allow for choice of different tools to create the desired end product.

1.3 Unity

Shacetha is implemented using the Unity (https://unity.com/) game engine. Unity is a cross-platform game engine developed by the Unity technologies and released in 2005, it allows for 2D, 3D, virtual reality and augmented reality software development using the C# language for writing scripts.

1.3.1 Game Engine

A game engine is large collection of code that is responsible for many basic components that make a game work, these usually include physics calculations, rendering, lighting and camera functions, this allows developers to quickly jump into creating game mechanics, level design, characters etc. without having to code every single detail themselves (Sinicki 2017).

1.3.2 Features and Stability

Unity is one of the most popular game engines (Toftedahl 2019; Itch.io 2021), this is due to two main reasons, first one is that Unity has been around since 2005 giving it a lot of time to improve its features, add new ones and gain engaged developers that share the knowledge through tutorials, course books etc. being on the market for this long also comes with the benefit of being able to pick a stable version such as the long term support (LTS) versions to ensure stable development environment (Unity Technologies 2019).

Second reason is the ease of use, Unity is very easy to start working with the different features present in the editor to modify the game world variables. It also features its own input management system, audio system, scene management, rendering pipelines and many other components that have proven very useful for game development and allow developers to focus on the design and implementation of game mechanics (Sinicki 2017).

1.3.3 Resources

As mentioned in section 1.3.2, Unity has a lot of great resources to help new and even experienced developers to create games, some of the resources include books, courses, online tutorials and discussion forums. These resources are a great way to learn game programming techniques, efficient code writing, managing a project and many other useful information about game development.

2 Designing And Prototyping

2.1 Development Environment

Developing software often involves using many technologies with different options for most problems encountered. This section gives a quick overview of the development environment and reasoning behind some of the choices.

2.1.1 Integrated Development Environment

Writing scripts for Unity applications can be done using different IDEs, the most popular and featured include Visual Studio (https://visualstudio.microsoft.com/) and JetBrains Rider (https://www.jetbrains.com/rider/). Rider has great integrations with Unity, allowing to control Unity's editor within the IDE, highlights code that has potential for reducing performance and how to improve it and a great integration with a lot of the Unity components including shaders, documents etc. Visual Studio doesn't have as much integration with Unity as Rider but has many useful features that make coding C# faster including suggestions and code highlights.

This project was written using Visual Studio there were two main reasons for this: first is that I am more familiar with Visual Studio allowing me to quickly set up and start developing without having to learn a new tool, second reason is that Rider requires buying a license.

2.1.2 Game Engine & Programming Language

When working with Unity all scripts are written using the C# programming language, for this project the Unity 2019 LTS version was used. These versions don't get any new major features but receive bug fixes, stability and performance updates. LTS versions are a great way of creating projects in Unity without having to worry about broken features that prevent making progress in the project. The choice of 2019 LTS over 2018 LTS comes down to few reasons. First reason is the universal rendering pipeline which increases performance and makes it easy to implement features like lightning in 2D. The most important reason for choosing 2019 LTS is that it introduced a lot of 2D feature sets and workflows like the mentioned lightning and improvements for tile-based world building which is the main way arenas in Shacetha are build.

Tilemaps are a great way of building 2D levels, making it easier to build levels but also greatly helps to improve performance reducing file size and RAM usage by reducing the amount of gameobjects created when making a level (Unity Technologies 2020).

The Unity tilemap system has many great features but it can easily be expanded by getting 2d-extras package from GitHub (Unity 2021), it provides new brush types to make the process of creating levels and/or worlds faster for example the animated brush allows to make tiles with animation which can make the world feel more alive, another very useful brush is the prefab brush which allows to quickly put prefabs on the map, any object can be converted to a prefab which is a way of creating a template from that object to allow easier adding into the world.

2.1.3 Version Control, Progress & Issues Tracking

Having a version control for software development is very important, with Unity projects the most popular choices are GitHub (https://github.com/) and Unity's own Collaborate (https://unity.com/unity/features/collaborate) service. GitHub is a well-established version control service with many features that make it easier for users to manage their projects, these include branches, tags, releases, tracking of issues and tasks. Main benefit of Collaborate is the ease of using it, as it is integrated into the Unity editor but the benefits of a service like GitHub make it the better choice in most cases and that is what was used for Shacetha.

To make the development process more smooth and to keep track of any bugs, issues and tasks a Trello (https://trello.com/) board was used. Trello allows the creation of a board with different sections such as to-do, done, issues and then create tasks that go into those sections allowing for better management of the work done and still to be done. GitHub also has a nice issue tracking feature which was used to keep track of any bugs that were found during development and/or playtesting.

2.1.4 Game Assets

All assets used in the game are either free to use or have been purchased by me in the past and are eligible to be used for this type of project i.e. non-commercial, academic, with the appropriate attribution if required. Aseprite (https://www.aseprite.org/) is a pixel art tool that also has great features for creating sprite animations and was used to create any additional sprites needed.

2.2 GAMEPLAY DESIGN

This section goes into the gameplay design choices, what the main gameplay loop in the game is, an overview of how the game controls and how the player progresses in the game.

2.2.1 Player Character

Players control one main character; the game is intended for keyboard and mouse using the keyboard for movement and user interface (UI) controls for example opening the inventory. The mouse is used for equipping, aiming and using their items, this can include swinging their weapon, blocking with a shield or other actions depending on the item.

All characters in the game including player and enemies have health points (HP), HP is a numeric value that keeps track of how much damage the character has received and once it reaches zero the character dies and in case of the player the game ends. Armor is a stat that is closely related to HP, all characters have a base armour value that will reduce the incoming damage by percentage value, this will never reduce the damage to below one, this armour value can be increased by certain rewards.

Another common component between all characters is the *HandsController* script, this script is responsible for equipping and using items in either of the characters hands. The hands move in 360 degrees around the character and the range of the action depends on the item type. In case of the player character the *PlayerController* script is responsible for handling input from the player and calling equip and use actions through the *HandsController*.

2.2.2 Enemy Characters

All enemies have stats like the HP values which can differ depending on the enemy i.e. enemy stats can differ from each other to provide more variety in the game. Another way that the enemies are different from each other is that they all come with pre-equipped items but depending on the items they have equipped their behaviour will be different, for example if they have long range weapons they will try to attack at a higher distance compared to an enemy with shorter range weapon.

While the player character is controlled by the player’s input, the enemies are controlled by a ‘brain’ script which is responsible for the behaviour of the enemy character i.e. how it moves and attacks, section 2.3.5 goes into more detail about how this brain script is implemented in Shacetha.

Bosses are stronger, more unique enemies that have at least one special ability that is often very dangerous to the player. Bosses are bigger to convey to the player that they are more dangerous and to make them more distinct from the other enemies. Boss abilities include spinning in a direction doing massive damage to anyone in the path, jump/charge towards the players location, grabbing a player if close enough and throwing them into the ground.

2.2.3 Items

Items are the main way players get rewarded in Shacetha, they also have great influence on how a character plays. All items have their own stats which include damage/armour, the item type, use cooldown, range and duration, this allows the creation of different items with unique stats but of the same item type to create more variety.

There are different item types i.e. swords, spears, shields, axes, daggers. All of these items types differ from each other in four areas:

- Range i.e. how far the item reaches in the game world.
- Area of Effect (AoE) i.e. what area the item effects.
- Modifier value of the item which in case of the weapons is the damage value and for shields is the defence value.
- Use cooldown which is the minimum wait time before being able to use that item again i.e. items with short cooldown can be used more frequently.

The different weapon types have different stats and behaviour which is summarised in Table 1, as an example a sword has all around average stats but can attack multiple enemies thanks to the big arc AoE, while a dagger is great at dealing huge amount of damage but in a small point and short range. Shields can be used to increase character’s armour for certain amount of time, by providing the armour only for short amount of time it make shields more interesting to use as the player has to use it at a time where they expect to be hit allowing more skilful gameplay.

Item Type	Range	AoE	Modifier Value	Use Cooldown
Sword	Medium	Big Arc	Average Damage	Average
Axe	Medium	Average Arc	Large Damage	Long
Dagger	Short	Small Point	Large Damage	Short
Spear	Long	Average Point	Average Damage	Average
Shield	n/a	n/a	Defence	Long

Table 1. Comparison of Item Types

The main reason behind several item types that behave differently is to allow players to equip any item into either of their hands. This allows them to choose the items that fit their playstyle, if someone prefers doing massive damage but being in constant danger they can go for daggers while someone less experienced with the game could go for a spear and shield to keep enemies at a distance.

2.2.4 Flow of the Game

The main mode of the game is the arena mode, in which the player is put in a level/arena and enemies start appearing or spawning from different points in the arena in waves i.e. a level might consist of five enemies and more won't spawn until all of the previous enemies are defeated. Every five waves a boss will appear, once the boss is defeated player receives a reward and continues with the normal enemy levels. The arena mode is endless which means it requires different ways of keeping the player engaged and feeling like they are making progress.

In addition to the methods in section 1.1.3 to keep players engaged i.e. keeping track of highest level and improving their own skill level, each run in Shacetha should feel different to increase replay-ability. The First way to achieve this is by having enough variety in enemies and items where when starting a new game. Players should be encouraged to try a different play style and use different items. To accomplish this replay-ability a good reward system has to be put in place which is discussed in more detail in section 2.5.

2.3 AI Design

Enemies are one of the core parts of Shacetha, therefore having interesting and challenging artificial intelligence (AI) controlled opponents is important to keep players engaged in the game. The enemies should have some predictability to allow players learn the enemies and improve their own skill level to have an easier time next time they encounter same enemy (Koster 2013). This section goes into more details about creating engaging AI controlled enemies and discusses two popular approaches to creating AI for non-player characters (NPCs).

2.3.1 Creating Engaging AI NPCs

With any game there is a certain level of challenge that is expected by players. Challenge that will let them test their skills, of course this can differ from game to game for example in a puzzle solving game the challenge comes from understanding the puzzles and how quickly someone can solve them. In a game like Shacetha the main challenge comes from the AI controlled NPCs i.e. the enemies (Sabbagh 2015).

One very important way to keep enemies engaging is to provide a way for the players to learn the opponent's behaviour patterns that help them in future encounters. One of the best ways of achieving this is for enemies to have a 'tell' which appears before an enemy attacks to which the player can react. These tells should be obvious and different depending on the attack to ensure the player can react correctly if they are quick and skilled enough (Vossen 2015). The tells can range from changing colour, pulsating sprites or even special indicators like an exclamation mark above enemies head, for Shacetha the main indicator used is altering the sprite in some way that is the sprite might change colour or start growing bigger and then going back to normal after the attack.

2.3.2 Finite State Machines

One of the choices for implementing the AI in Shacetha were Finite State Machines (FSMs) which were arguably one of the most popular approaches to creating AI NPCs in video games for some time. Many games like Doom, Quake and Half-Life used the FSM approach (Sweetser and Wiles 2002). In FSM a game object can be in one of the possible states with transitions in and from those states. An example of FSM can be seen in Figure 2, an opponent character can be in one of the four states i.e. Idle, Searching, MoveToTarget or Attack and can change state by one of the available transitions for example if the enemy is searching it could start moving towards their target if they find one in their sight range.

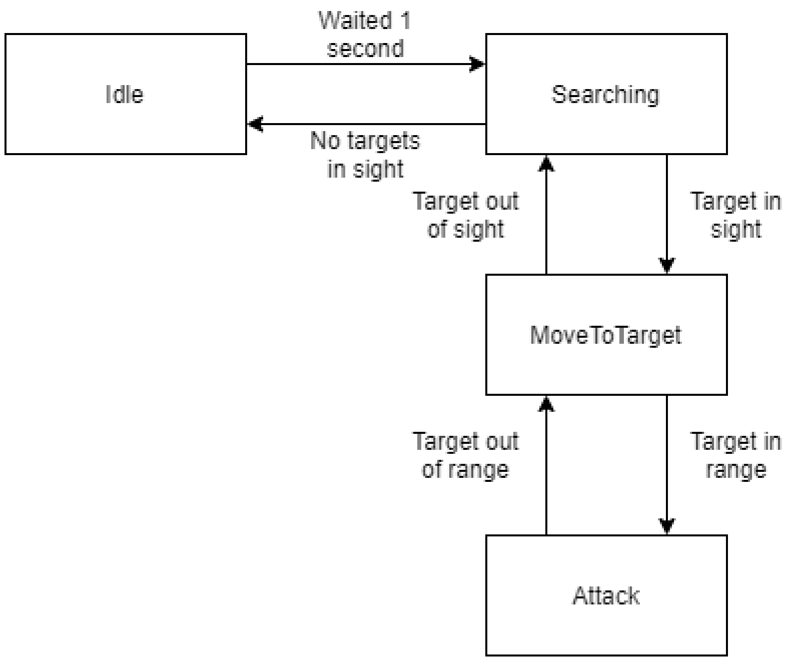


Figure 2. Example of a Finite State Machine

FSMs are simple to implement, provide a lot of power relative to the complexity. They are also very easy to understand and debug which is very helpful when creating new FSMs making it very clear what the desired behaviour is. These advantages can be neglected by a poor structure i.e. FSM in games can often include states within states, many variables within a state, code that is being executed every frame/tick all of these make FSMs difficult to test, expand and maintain (Sweetser and Wiles 2002; Bevilacqua 2013). Another major advantage is that FSMs are computationally inexpensive, making them great for video games where there are already a lot of systems working together that can be difficult for some machines to handle.

Some of the main disadvantages with FSMs are that they scale very poorly, this is due to the fact that adding new behaviour to the system, requires adding the new state and then adding all transitions to and from that state. This process can be very difficult when dealing with a very large system with hundreds of states and is prone to errors. Another major disadvantage is that FSMs are very deterministic which can limit what can be done with them i.e. anything implementing the FSM has to be in one of the possible states and all transitions are pre-determine. This means that after some time all FSMs become predictable in what they will do next (Sweetser and Wiles 2002; Bevilacqua 2013).

2.3.3 Behaviour Trees

Another choice for implementing AI was behaviour trees (BTs). BT is a hierarchical tree that is responsible for the decision making of an AI agent. The leaves which are the lowest nodes are also known as *action* nodes, as they are responsible for commanding an agent to do something for example attack, hide or heal. Leaf nodes usually connect to *control* nodes, that dictate the flow of the tree i.e. which action node should be executed. An example of a BT can be seen in Figure 3, in this example the *control* nodes are *Repeat*, *Selector* and *Sequence* while *action* nodes are *TargetInAttackRange*, *Attack*, *TargetInSearchRange*, *MoveToTarget* and *Idle*. Nodes are executed left to right, *Repeat* just means the tree is repeated. The *Selector* node will execute until a node returns success or all nodes fail, and a *Sequence* node will execute until a node returns failure or all nodes execute successfully. A summary of the nodes used can be seen in Table 2. In the BT seen in Figure 3, the AI agent will first check if their target is in attack range and if so it will attack them, if that resulted in failure it will check if target is their search range and move towards them if that returns success, if attack or movement fails it will stand idly.

Node Name	**Node Type**	**Behaviour**
Repeat	Control	Repeats execution of children
Selector	Control	Executes until first success or all failed
Sequence	Control	Executes until first failure or all success
Probability	Control	Picks a child node based on assigned probabilities
TargetInRange	Leaf	Returns success if target is within given range
Attack	Leaf	Returns success after executing an attack
MoveToTarget	Leaf	Returns success after finished moving
Idle	Leaf	Returns success
Inverter	Decorator	Returns opposite result of its child node

Table 2. Summary of Planned BT Nodes

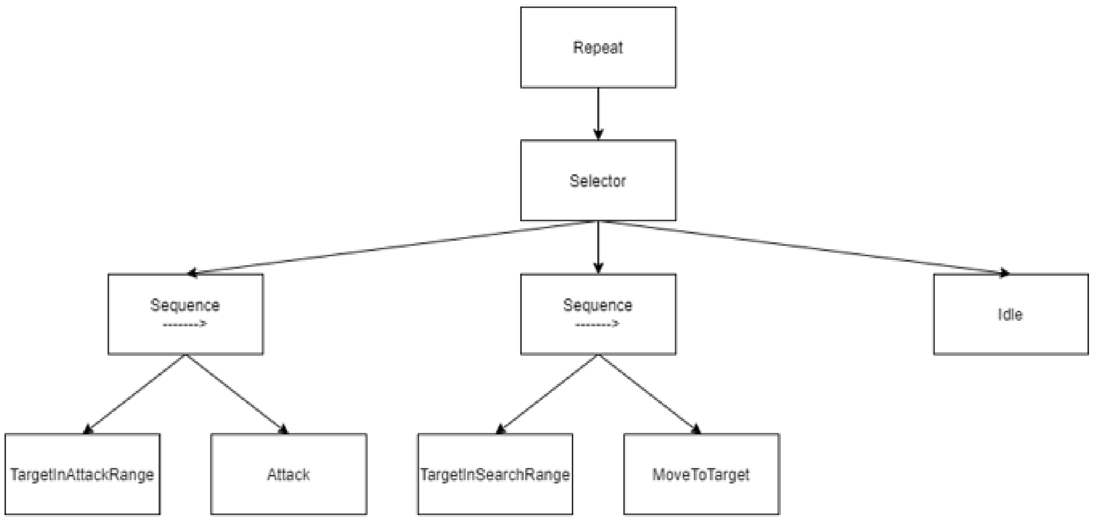


Figure 3. Example of a Behaviour Tree

BTs are simple to implement, are very easy to understand and debug. A BT can start with very basic behaviour such as movement and attacking, later this can be expanded by adding more nodes at the correct branches. For example a boss enemy can use the basic tree and add a sequence node that checks for ability cooldown and uses the ability when it is not on cooldown (Simpson 2014), this would be added before the attack section as seen in Figure 4. BT are also modular which greatly increases reusability, allows for nodes to run in parallel when required and finally it is possible to add some unpredictability thanks to *Probability* control nodes which allows them to select a node based on a random value, this allows the creation of more unique AI agents that seem smarter as they won't always behave the exact same way (Lim *et al.* 2010; Simpson 2014).

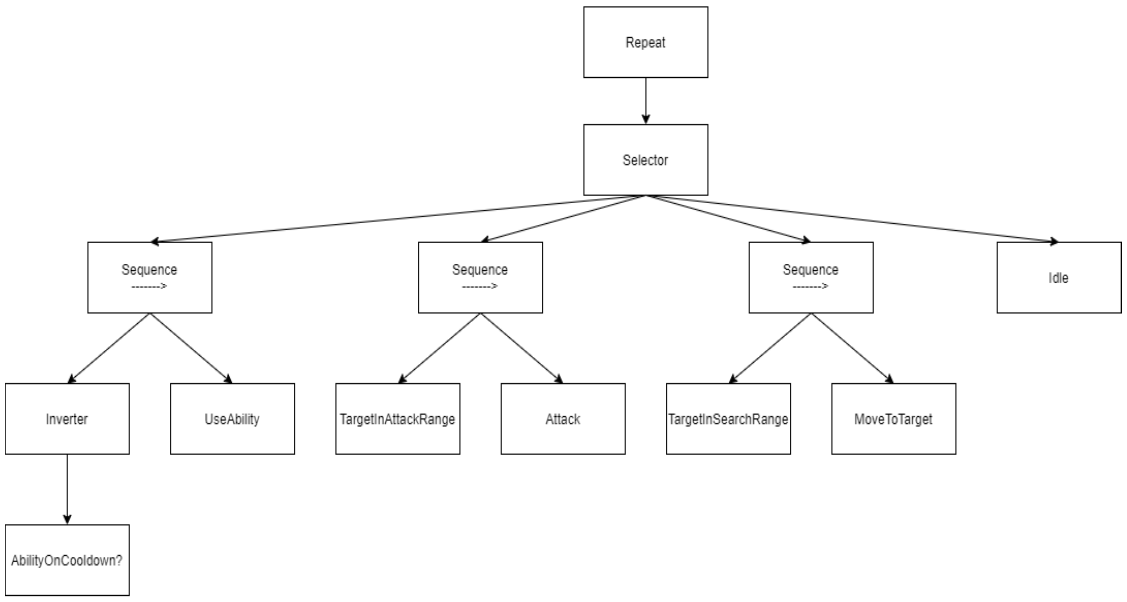


Figure 4. Example of an Expanded Behaviour Tree

BTs have some disadvantages, as the tree grows the computation cost of going through it also grows which can cause problems for games with very complex AI agents (Rasmussen 2016).

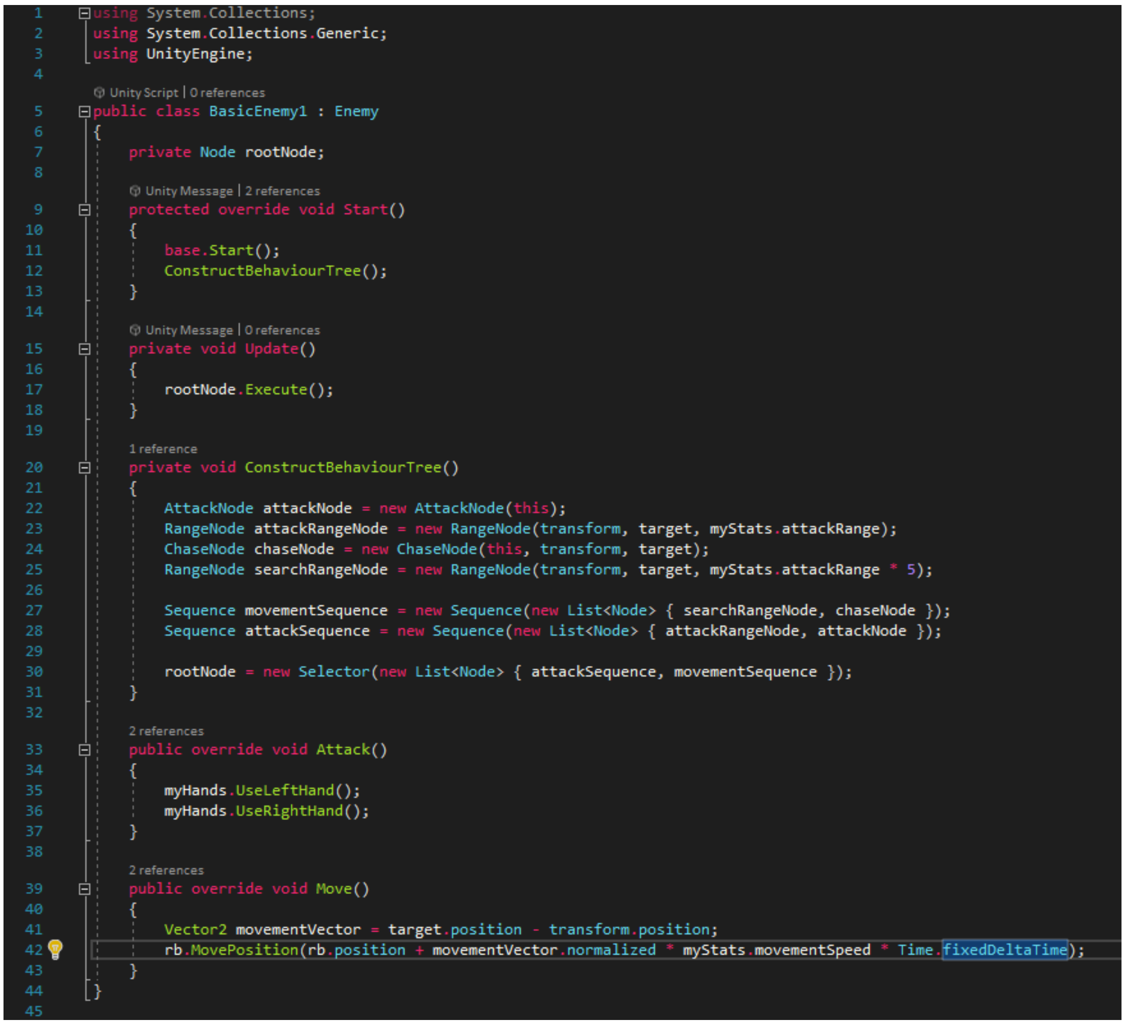
2.3.4 Summary

While both FSMs and BTs are simple to implement and would work for Shacetha as the AI agents will not have a huge amount of nodes/behaviours. BTs provide greater flexibility with their *Probability* nodes and are easier to expand on, as adding nodes to the tree is easier than having to add all possible transitions when adding a state to a FSM (Lim *et al.* 2010). While all enemies in Shacetha are meant to have some predictability to allow players to learn their patterns, the main way to achieve this are the tells mentioned in section 2.3.1. By having some enemies that randomly choose between actions, it can create more interesting fights that a player can still learn and adds to the replay-ability of the games thanks to some enemy behaviour variance.

2.3.5 Implementing BTs

Implementing BTs in Unity is not too complicated, the first step required is the creation of a base node class i.e. node is the basic building blocks of a BT, the next step is the *control* nodes which include *Selector* and *Sequence,* there is no need for a *Repeat* node as the tree will be run every time the AI agents update method is called. Then there are the *action* nodes which include the ability, attack and chase nodes. The *Range* node which will return success or failure depending if the distance between two given transforms is less than the supplied threshold, transforms can be any object's position including enemies and player. The last node is an *Inverter* node which inverts the child return value i.e. if the child returned true the inverter will change it to false and vice versa.

Once the required nodes where implemented now it was time to create the trees in the agent scripts. Constructing the tree is done in the Start method which is called when a script is enabled and before calling the Update method. Building the tree starts with creating the leaves or actions first, the next step is creating the control nodes with the required action nodes passed into their constructors. Once all control nodes are created, they are passed into final control node constructor which is the root node. The root node is executed every frame in the Update method, this whole process can be seen in Figure 5. As the trees are small enough executing them every frame should not cause any issues, but a delay or check can be easily implemented if it causes any major performance issues.


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 0 references
public class BasicEnemy1 : Enemy
{
    private Node rootNode;

    Unity Message | 2 references
    protected override void Start()
    {
        base.Start();
        ConstructBehaviourTree();
    }

    Unity Message | 0 references
    private void Update()
    {
        rootNode.Execute();
    }

    1 reference
    private void ConstructBehaviourTree()
    {
        AttackNode attackNode = new AttackNode(this);
        RangeNode attackRangeNode = new RangeNode(transform, target, myStats.attackRange);
        ChaseNode chaseNode = new ChaseNode(this, transform, target);
        RangeNode searchRangeNode = new RangeNode(transform, target, myStats.attackRange * 5);

        Sequence movementSequence = new Sequence(new List<Node> { searchRangeNode, chaseNode });
        Sequence attackSequence = new Sequence(new List<Node> { attackRangeNode, attackNode });

        rootNode = new Selector(new List<Node> { attackSequence, movementSequence });
    }

    2 references
    public override void Attack()
    {
        myHands.UseLeftHand();
        myHands.UseRightHand();
    }

    2 references
    public override void Move()
    {
        Vector2 movementVector = target.position - transform.position;
        rb.MovePosition(rb.position + movementVector.normalized * myStats.movementSpeed * Time.fixedDeltaTime);
    }
}
```

Figure 5. Constructing a BT in an Enemy Script

2.4 USER INTERFACE

When creating video games it is important to have a clear user interface (UI) that is helpful to players. The UI is one of the easiest ways of causing players frustration and confusion, as if the UI is annoying or cluttered it can quickly discourage a player from trying out a game. UI should not take away the players focus from the main game mechanics and should be clear what each part is for without needing long tutorials or documentation. In this section I got into a bit more detail about what makes a good UI and the design used in Shacetha.

2.4.1 A Good User Interface

When creating a UI for any software it is important to create a design that contains the relevant and required amount of information, this is to ensure the user can easily understand what they are looking at and not be distracted by extra clutter or completely alienated by the UI (Guntupalli 2008). Another important part of a UI in video games is that they should not break the player's immersion in the world where possible, this can range from trying to keep the UI elements as part of the game world to having the menus match the overall atmosphere and aesthetic of the game (Stonehouse 2014).

2.4.2 Designing & Prototyping UI

When it came to create the UI in Shacetha first step was to draw the design on paper, it helps to visualise what is required before committing any time to prototyping and coding in Unity. It was important that the screen doesn't get too cluttered and that all elements have a purpose, as can be seen in Figure 6 there are four parts to the UI. The Bottom left shows the players HP value so that they can always be aware of how close to death they are. The bottom centre shows the left and right actions ensuring the player is always aware of what they have equipped and when implementing a prototype in Unity a cooldown indicator was also added i.e. if an item was used it will have an indicator to show when it can be used again.

The other main parts of the UI are the character sheet and inventory. The inventory shows items owned by the player which they can equip in either hand, the left and right action keys are used to equip the item into the correct slot i.e. if left clicking on an item it will equip in the left hand and right click will equip in right hand. Another part of the inventory are tooltips that will display the items stats so it is easy for the player to see what they are equipping and to compare stats with what is currently equipped. The second part of the UI is the character sheet. This just gives a summary of the stats and shows the currently equipped items allowing players to hover over them for a tooltip with the item's stats. Tooltip is just a small window with text which can been seen in Figure 8, which will contain the item name and stats. Both these windows have only the necessary information required without any extra clutter to make it easier to understand.

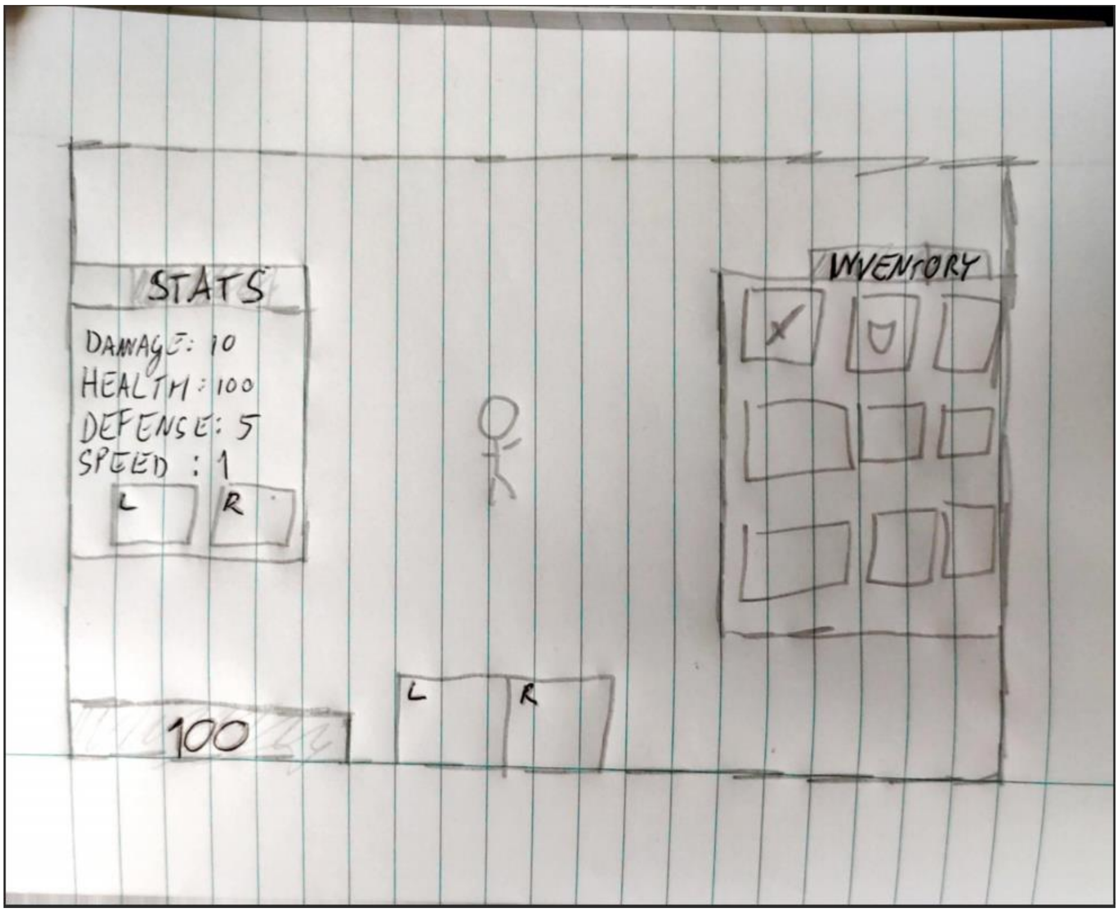


Figure 6. UI Paper Prototype

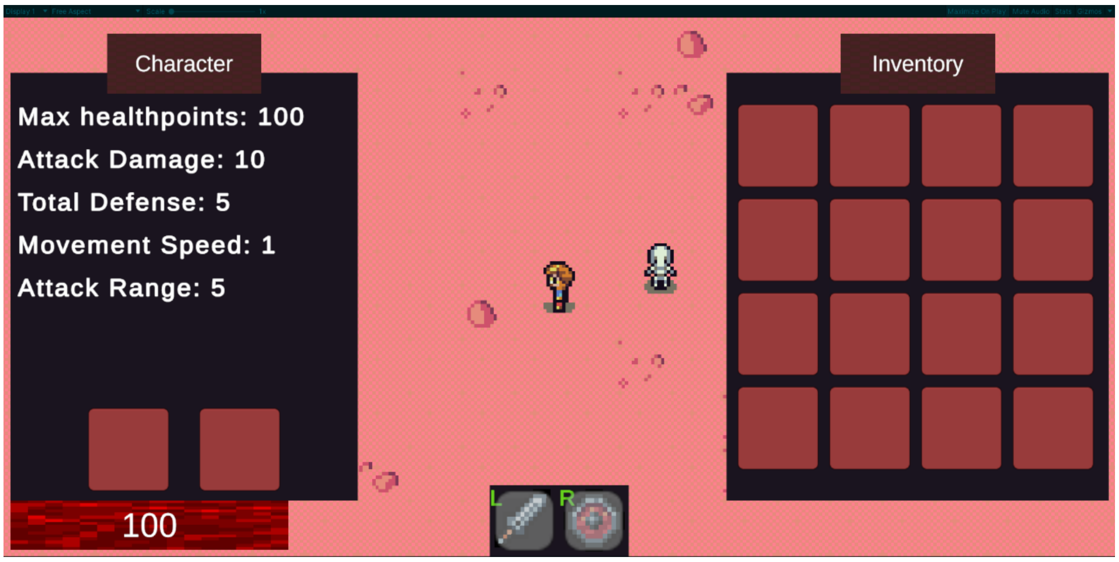


Figure 7. UI Unity Prototype

2.5 REWARD SYSTEM

Reward systems on their own will not make any game perfect but they are a great way of keeping players engaged in the game and wanting to continue playing it. This point is supported by many other products trying to use the process of “gamification” i.e. adding some form of reward structure such as points, to keep people engaged with the product (Koster 2013).

For Shacetha the plan is to have two main components in the reward system: first is a point system by having the game keep track of how many waves the player has defeated, this allows players to have a goal to try to achieve i.e. beating their previous score or beating a score someone else achieved. The second is an item granting system which is widely used in the RPG genre of games, these systems are great at keeping players engaged with the game acquiring more items giving them more choices in the playstyle and collecting rare items to add to their collection (Wang and Sun 2012).

2.5.1 Choice of Three

The main idea for the reward system in Shacetha is a choice from three possibilities every two or three rounds and after each boss fight. The first two choices will always be items while the third choice will be some stat upgrade, this can be seen in Figure 8 in which the third choice is plus five to the defence stat. By having a combination of items and stat rewards it allows players to customise their character to fit their preferred playstyle. The main difficulty will come from balancing the item choices to ensure they don’t repeat which would limit the actual choice.

The rewards after a boss fight will be of greater power this means that the items will have better stats and the stat reward will have more of an impact. One way to make the reward choice more interesting is by having a downside to some rewards, for example a player might be able to pick a huge damage stat reward but it will come with reducing the max health points, this way it will provide players with a meaningful choice in how they want to play the game. One player might choose to pursue full damage hoping they can avoid most attacks while another could go for the slower approach with more defensive stats.

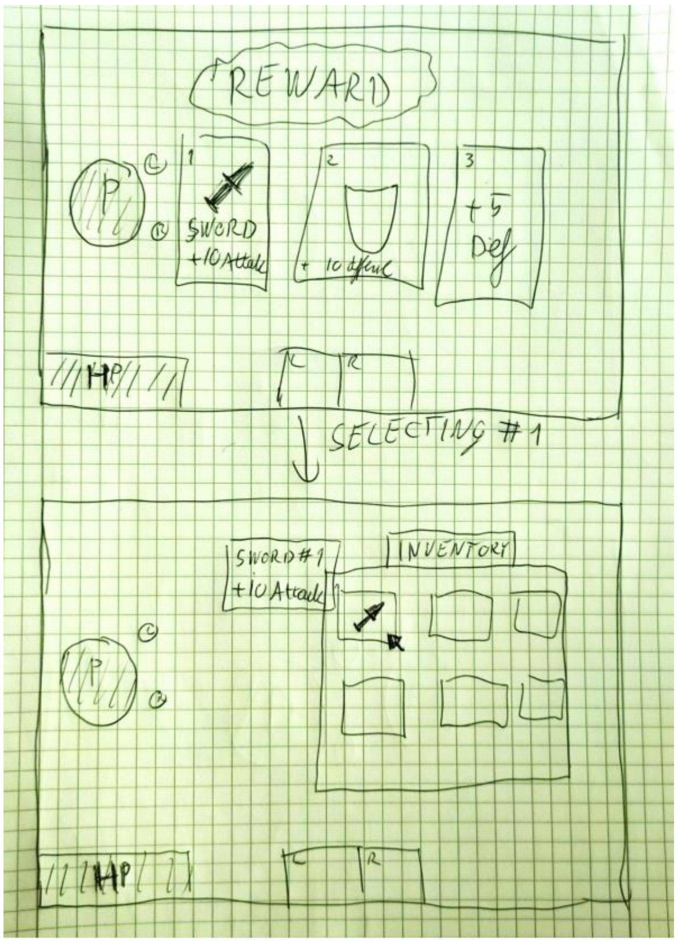


Figure 8. Reward System Paper Prototype

2.6 Endless Arena mode

The endless arena mode will be the main way of playing Shacetha, it should provide players with engaging fights, interesting reward choices and a way for players to improve their own skill level by having enemies with tells that the players can learn.

2.6.1 Arena 'Manager'

The most important component for the arena mode is going to be the arena manager. This manger will be responsible for handling the different waves of enemies, which enemies should be created/spawned, when and what reward to give to the player. The manager will be a script attached to an object in the scene that won't have visual representation but rather work in the background. It will be important to fine tune the values in this script to ensure the game is balanced but also that the player feels like the game is challenging enough and gives interesting rewards.

2.6.2 Infinite Arena and Replay ability

The arena mode is infinite i.e. it will keep increasing the difficulty as the player reaches higher levels and will not end until the player is defeated, with these type of game mode it is important to add different ways to increase replay-ability as players can quickly become bored. The main way to increase the replay-ability in a game like Shacetha is by having a large variety of items, enemies and bosses, this allows each run or try of the mode to feel different by having different choice of rewards/items and fighting different enemies.

2.7 STORY MODE

There is no full story mode planned for Shacetha, the main focus of the game is on the arena mode but adding a one or two level tutorial would be important as an introduction for new players so they are familiar with the mechanics of the game and by adding a small story to the tutorial that takes players through the different controls it can make it more interesting rather than just showing the inputs on the screen.

3 INITIAL CHALLENGES IDENTIFIED

3.1 REWARD SYSTEM

3.1.1 Shop

One of the initial designs for Shacetha involved a shop system in which the players could earn some currency after each fight e.g. gold coins and would be able to spend them on items every few rounds. This would have given players constant rewards as they defeat enemies and better choice of items as the shop would have a range of items to buy but it quickly came clear that such system would require a lot of work to ensure fairness. One major issue is that it would be very important for items in the shop to have reasonable price for the power of the item, which in itself can be difficult to judge and would require a lot of play testing. If any item had their stats changed, it would also mean it would have to change in the shop. Another issue is that the shop itself would take large chunk of work to implement as it would require multiple UI screens and implementing the currency. Overall, these challenges resulted in the decision to de-scope the shop idea from the design and decided on a different reward system that is discussed later.

3.1.2 Random Rewards from Enemies

In addition to the shop another idea for the reward system was to have random rewards from killing enemies and bosses. This had a benefit of having the possibility of dropping an item whenever players defeated enemies which can feel exciting and fun. This system would require some form of player protection to ensure that items drop often enough so that the player doesn’t keep fighting without rewards. Even with this safety system the main drawback of purely random drop systems is that it leaves players with very limited control over their character, they can pick what items they want to equip but very little choice when it comes to what items they actually have.

Another issue with this system is that it limits the rewards to items and could result in repeats which could be prevented with extra functionality that would need to be designed, developed and implemented in code. This just highlights that this system needs a lot of safety measures and fine tuning values to ensure players get rewarded frequently enough with unique items.

3.2 Balancing

The balance of a game is how easy or difficult the different aspects of a game are, to give an overall challenging but fair experience. A game that is too easy will often result in players getting bored and moving on to a different one, while a game that is too difficult can seem impossible and frustrate players causing them to abandon the game completely, as supported by flow theory it is important to keep players in control of their actions and feel that these actions are impactful in deciding the outcome of an encounter (Jacko 2009). Some games thrive on the fact they are near impossible to beat but they have many other aspects that keep the player engaged. For Shacetha like many other games it is important for the game to be somewhere between the too easy and impossible sides, most people enjoy a challenge that is just hard enough to require some work and learning (Goodwin 2016; Sinicki 2017) which Shacetha hopefully achieves.

3.2.1 Scriptable Objects

Balancing can be a tough task that requires a lot of playtesting and fine tuning values to get the desired result. One of the ways to help with the process in Unity is to use scriptable objects (https://docs.unity3d.com/Manual/class-ScriptableObject.html). Scriptable objects are scripts that are subclasses of ScriptableObject rather than MonoBehaviour. MonoBehaviour is the default class all scripts in Unity inherit from, it includes methods that allow all objects to be created, updated and interact with the game world while the game is running, some of these methods include Start, Update and OnCollisionEnter2D. Scriptable objects on the other hand don't have these methods allowing them to be more lightweight and also allows multiple objects to keep reference to the same scriptable object instance for keeping track of stats (Hipple 2017). This is the main way they are planned to be used in Shacetha i.e. all items and characters contain a reference to either an item or character stats scriptable object.

Having all items and characters use scriptable objects for their stats provides a way to update all instances of the item or character by editing one scriptable object, furthermore this is a great method to fine tune the stat values without having to worry about forgetting about a single enemy instance hidden somewhere in the scene or prefabs. Another benefit is that because the scriptable object contains the stats, other scripts can reference that object to obtain stats of the entity without having to create specific methods for retrieving data from it.

3.3 PLAY TESTING

When creating any software, it is important to test it, for games it is important to play test them frequently, to ensure new features are working and fit with the rest of the game i.e. integration testing. It is also important to try to have people other than the developer test it, as it might often happen that the developer thinks a feature is boring or very easy because they have done it so many times, while someone completely new might have a different opinion. As such for Shacetha most of the play testing was done by myself but some of my friends and family have also played Shacetha and provided feedback. The project is also public on GitHub (https://github.com/) and Itch.io (https://itch.io/) which resulted in occasional plays and feedback from people on the internet. Any feedback given was considered but it was important to remember not all of it had to be completely true/correct, in the end it must fit the overall design of the game. While the project's source code is public no contributions were accepted.

3.3.1 Finding Bugs

While play testing it is important to keep track of the bugs encountered using GitHub or Trello to ensure they are fixed before completing the project, all bugs should be written with enough detail to clearly understand what the problem is and in what situation it has occurred with as much detail as possible to be able to re-create it.

3.4 POLISHING THE GAME

Once the main components of the game are finished it was time to ensure they are working as intended but it was also important to add some more items and enemies to ensure the game has enough content for players to enjoy it. This new content was restricted to items and enemies as they do not create major bugs, and the main parts were already be bug free but it was important to make sure they are balanced and fit the rest of the game.

3.4.1 More Items and Enemies

All of the item types have been implemented and there are different variants in the game for the player to use. It is important to have enough different items i.e. with different looks and stats to ensure the player has a good range of choice of rewards, the main characteristics of the item types can be seen in Table 1. While it is key to add more items there is a certain limit, as with too many items players might be stuck getting items they don't want, for this reason it is important to not just keep adding more and more items.

Another significant part was to add more enemies, this was mostly to increase the variety in the game. Some enemies might have same behaviour as previous enemies but have different stats and attacks i.e. a skeleton with a sword has less damage than a skeleton with a spear and their attacks differ.

3.4.2 Fixing Bugs

When adding new features bugs are expected. It is important to keep any software bug free, bugs can very often result in reducing the enjoyment from video games especially if they prevent the player from continuing playing or if they make them feel that they have unfairly lost due to issues in the game as such it was important to keep track of any issues found during play testing to fix before the final release version.

3.4.3 Adding Music and Sounds

Music is a great way to enhance atmosphere of a game which helps players more immersed in the game which results in overall better experience. Sounds are another effective way to improve the atmosphere of the game but also are great way to give players feedback of an action for example good attack sounds can create a more satisfying combat system (Vossen 2015).

4 IMPLEMENTATION

4.1 GAME SYSTEMS

4.1.1 Managers

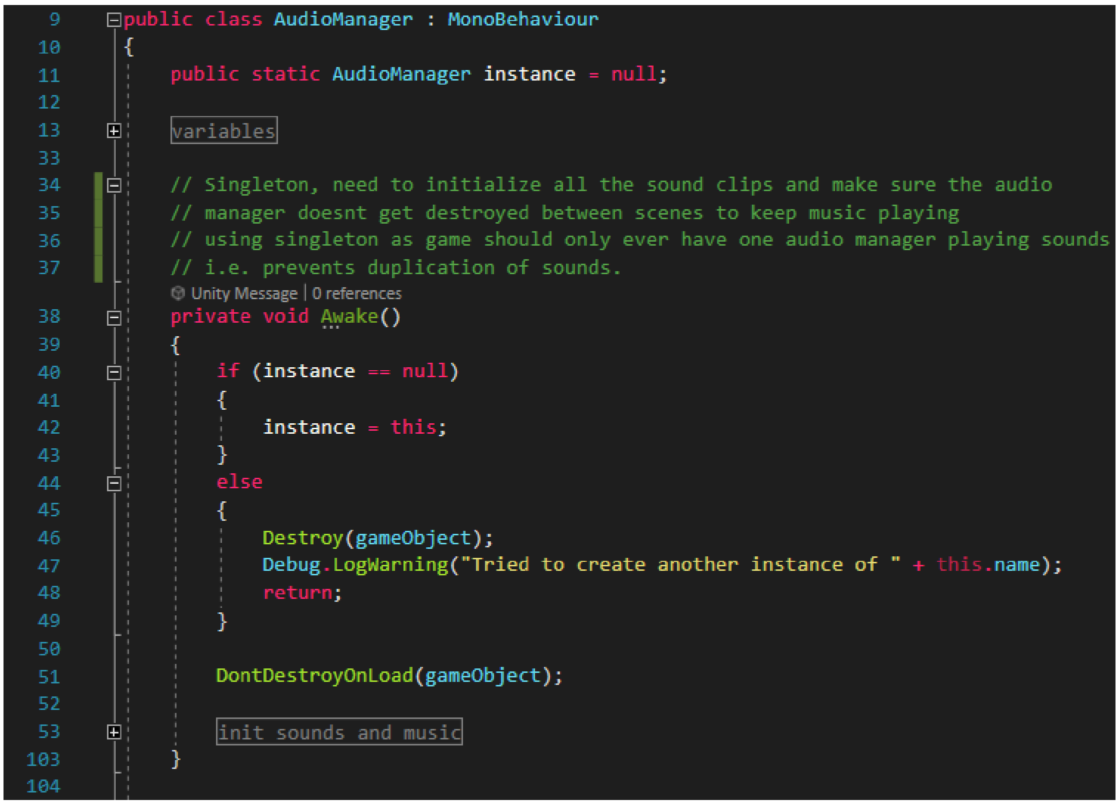
Shacetha includes Managers that handle different aspects of the game, these include the *AudioManager*, *PlayerManager*, *ArenaManager* and *RewardManager*.

AudioManager

The *AudioManager* is responsible for handling any audio related functionality, storing and playing all the sounds and music found in Shacetha. It uses the singleton pattern which ensures there is only ever one instance of it and providing a way to access that instance for other classes (Bond 2017). By only having one instance of this manager it prevents a potential issue of multiple sounds being played at the same time. Another benefit of singleton is that the instance can easily be set to not get destroyed between scenes, this improves performance as no new object is created. Finally, it allows for music to play in transition between scenes without interruption, scenes are different levels created in Unity which can be loaded through scripts when needed. This is done using the Unity's DontDestroyOnLoad function

(http://www.docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html).

Using a singleton pattern allows other scripts to find the instance even when moving to a new scene, this is achieved by having the instance of the class be static. An example of how a singleton pattern looks in code can be seen in Figure 9. Other scripts can use the *AudioManager* instance to request required sounds to be played as needed, for example swinging sounds when a weapon is used, damage sounds when a character takes damage or playing boss music during a boss level.



```
public class AudioManager : MonoBehaviour
{
    public static AudioManager instance = null;

    variables

    // Singleton, need to initialize all the sound clips and make sure the audio
    // manager doesnt get destroyed between scenes to keep music playing
    // using singleton as game should only ever have one audio manager playing sounds
    // i.e. prevents duplication of sounds.
    Unity Message | 0 references
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
            Debug.LogWarning("Tried to create another instance of " + this.name);
            return;
        }

        DontDestroyOnLoad(gameObject);

        init sounds and music
    }

```

Figure 9. Example of Singleton

PlayerManager

The *PlayerManager* oversees functions related to the player character. It is a way for enemy scripts to quickly access the player script to deal damage. Another use for this manager is to stop and resume input for the player when pausing the game. Lastly it is used to display a death screen to the player once they are defeated and reset the player character state i.e. setting player's stats back to the default values when starting a new game. This manager also uses the singleton pattern as there should only ever be one player object in the game world.

ArenaManager

The *ArenaManager* script is responsible for keeping track of the current level, creating wave of enemies or bosses depending on the level, change the arenas every ten levels and keeping track of enemies killed to raise an event to open the reward screen which is controlled by the *RewardManager* script. The *ArenaManager* is also responsible for updating the highest level a player has reached.

RewardManager

Finally, the *RewardManager* controls the logic of picking and presenting rewards to the player. The reward screen is shown either after defeating a boss level or after round ends with the player having defeated four or more enemies since their last reward. The RewardManager picks the rewards based on pre-defined probabilities, which can be seen in Table 5 found in Appendix A.

4.1.2 Event System

Shacetha includes an event system that is implemented using scriptable objects. This event system works by adding a *GameEventListener.cs* script which was created for this project, to any object in the game world and registering a game event that this listener will respond to, for example arena manager listens to enemy died event and calls the *EnemyDied* function when the event is raised. All events are instances of the *so_GameEvent.cs* script which is a scriptable object script created for this project. This allows the creation of different instances or events without having to write multiple scripts. The main way this event system is utilised in Shacetha is to signal when an enemy dies, which is used by the ArenaManager to keep count of how many enemies player has defeated. These events are also used to raise an event when the player dies, which is used to perform any functions before the game resets for example to check if the player has reached their highest level of arena and if that is the case save it. This system is also very useful when needing to pass messages between different scripts for example the ArenaManager raises a reward event which the RewardManager listens for to calculate new rewards and present the reward screen to the player.

4.1.3 Other

Another important system found in Shacetha are tooltips. Tooltips are displayed when a player moves the mouse cursor over an item in their inventory or in the reward screen. This is very important to allow players to make an informed choice, by having the tooltips it allows players to compare items and choose what they think will be best for their playstyle and situation. Some games hide such information from the player to add mystery to the game, allowing players to experiment and discover parts of the information by trial and error. For a game like Shacetha it was always more important for users to have this information available to allow them an easier way to get the items that fit their playstyle.

The inventory system in Shacetha allows players to open a window that shows all the items, allowing to mouse over them for the tooltips with stat information and use left or right mouse buttons to equip the item. Lastly there is a fire button in top right of each inventory slot to allow players to destroy an item in case they require space for a new reward.

4.2 REWARD SYSTEM

4.2.1 Overview

The reward system has been implemented as planned in section 2.5, with some slight alterations which are discussed further in section 4.2.2. The reward system provides two item choices which are chosen from a list of 29 items of which, 5 are epic, 10 are rare and 14 are normal rarity. Stats for all of the different items can be found in Table 7 and the probability of a given item rarity can be seen in Table 5, both can found in Appendix C. The third reward is a stat reward that can give a positive or negative to one or two stats. Items also have a chance to provide an extra stat reward which can be positive or negative, this tries to make the choice of an item more interesting, an item can be powerful but could come with a drawback of a negative effect to one of the stats.

4.2.2 Challenges Encountered

From playtesting one issue that became very clear is that player's character would often end up taking a lot of damage and no way to reverse it. This made the game very punishing and often reducing the enjoyment from the game. To fix this issue two things were added to the reward system. All rewards now restore five HP and a fourth reward choice was added, this choice always heals the player for a quarter of their max HP. Another solution that was considered was to restore player to full HP after each round, but this would make taking damage in a level feel very insignificant as long as the player didn't die.

Adding the fourth choice to the rewards was a good solution that reduced the punishment from taking a lot of damage in a round without completely forgiving players for getting hit. This also provided players with an interesting choice in which they could ignore the health reward to get a powerful item even if it would mean that they would be very close to dying, which is a great example of risk versus reward i.e. picking something with greater risk for a greater reward (Lambottin 2012).

Another challenge found was that the reward system could give a choice of a stat reward which gave same positive and negative reward to the same stat, for example plus four and minus four to damage, this would result in a neutral reward. This was fixed by preventing the RewardManager from picking same stat twice in the reward picking.

4.3 GAMEPLAY

4.3.1 Combat

The combat system in Shacetha went through a major rework during the implementation phase. In the first system the combat was based on using Unity's physics engine OverlapCircle method

(https://docs.unity3d.com/ScriptReference/Physics2D.OverlapCircle.html).

This worked by creating an invisible circle in the game world that returns all objects that were inside it, and then all objects that were enemies would get attacked and damaged. It functioned effectively for most scenarios, but the main drawback was that the attacks only ever did damage when the circle was created, and for the rest of the weapon's swing it would do nothing. This required a lot of timing to ensure the circle was created at the correct time in the weapon's swing animation as otherwise it would seem like the weapons has done nothing.

In the second iteration of the system the combat system now uses colliders, which are a component added to an object for calculating any collisions in the world and can be disabled until the weapon is used. Using Unity's animation system allows the creation of different animations depending on the weapon type to create different behaviour for all weapons types. Once a weapon is used and it does it's swinging animation the collider is enabled and can have an effect on all object it touches. This allows for weapons like the sword and axes to do damage in greater areas as the collider now can follow the weapon during its animation. One drawback to this system is that while the collider is enabled it can damage same enemy multiple times, but this was quickly fixed by giving all characters invincibility timers after taking damage, during which they cannot take damage.

Another addition found in this version of the system is that all characters are now unable to move during their attack and the time they must be stationary depends on the weapon's cooldown. This allows to make more powerful weapons have a longer cooldown to introduce a drawback of being stuck in place for longer when using them. This makes using items have a risk and reward aspect to them. For example, slower weapons require more strategy and planning as a badly timed attack can result in taking damage due to being trapped in place, but landing an attack with one will result in massive damage (Lambottin 2012). This helps to enforce the idea of having to learn the enemies attack patterns to use items effectively.

With this combat system, there is a formula for reducing incoming damage by the total armour value a character has. A graph showing the result of the formula can be found in Figure 10, the x-axis is the total armour a character has, the y-axis is the percentage of damage that will be done to the character. From the formula we receive a curve which means that to a certain value i.e. around 200, armour is very effective at reducing the incoming damage but as the values get larger the gain from armour is reduced. This helps to prevent players from trying to get very high amounts of armour to receive no damage and allows for better balancing of stats for both characters and items to create more satisfying combat system.

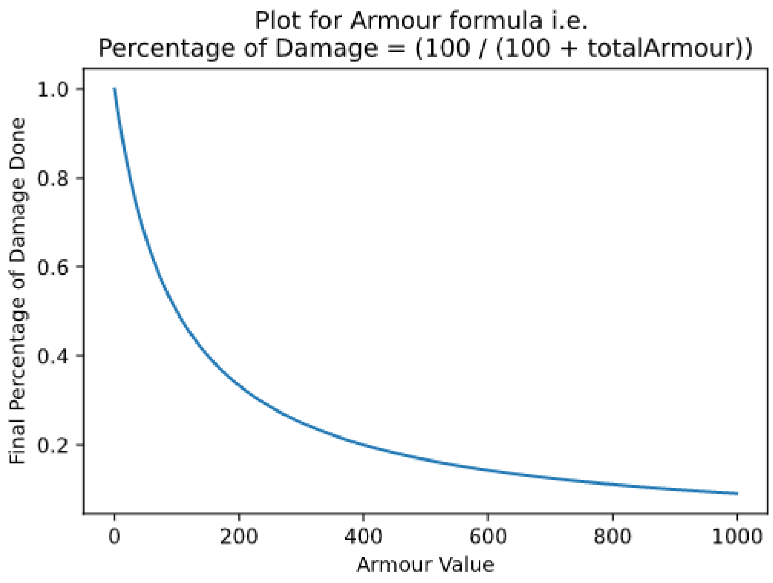


Figure 10. Graph of Damage Percentage against Armor Value

4.3.2 Arena

As planned in the design stage discussed in section 1, the endless mode takes place in an arena, has infinitely spawning enemies and increases in difficulty as the player reaches higher levels. Shacetha also includes four different arenas that differ by their looks i.e. different sprites used for the tile maps to have different ground and decoration textures. Another difference between the arenas is the traps that are present in them. Three of the four arenas include spike traps that activate when player walks over them and only damage the player character. These spike traps are placed differently in all of the three arenas to create more danger areas for players to avoid. The second arena contains traps that shoot projectiles, this adds a constant risk of taking damage and keeps players moving to avoid taking unnecessary damage. Arenas switch every ten levels up to level 31 where the final arena is used for the rest of the levels. These different arenas are a great way to add variety to the endless game mode and can provide new and different challenges thanks to the traps in them.

The creation of enemies is handled by the ArenaManager and has gone through different versions before the final one. Arenas have multiple spawning points for basic and boss enemies at which the enemies can be created, a position is picked randomly by the manager from these spawning points. Initially the number of enemies created was based on the current level and it always picked from the same list of enemies.

In the final version the manger has three different lists of enemies separated based on their difficulty and increases number of enemies based on how many enemies it has made already. Once the creation position has been picked it then calculates how many enemies should be instantiated. The game starts off with one easy enemy being created and increases that by one each level. After reaching five easy enemies it will start using medium enemies and once it reaches three medium enemies it will begin using hard enemies. For bosses it will use medium bosses after three easy bosses and hard bosses after two medium bosses. Table 3 shows the count for how many of each enemy difficulty will be created in the first twenty levels. Lastly once the number of enemies has been calculated it will randomly pick an enemy from the corresponding difficulty list and create it at the chosen location. The enemy is picked randomly each time to prevent the manager from creating only one type of enemy in a level.

Level	Enemy Type	Easy Enemies	Medium Enemies	Hard enemies
1	Basic	1	0	0
2	Basic	2	0	0
3	Basic	3	0	0
4	Basic	4	0	0
5	Boss	1	0	0
6	Basic	0	1	0
7	Basic	1	1	0
8	Basic	2	1	0
9	Basic	3	1	0
10	Boss	2	0	0
11	Basic	4	1	0
12	Basic	0	2	0
13	Basic	1	2	0
14	Basic	2	2	0
15	Boss	3	0	0
16	Basic	3	2	0
17	Basic	4	2	0
18	Basic	0	3	0
19	Basic	1	3	0
20	Boss	0	1	0

Table 3. Summary of Enemies Created in the First 20 Levels

4.3.3 Tutorial

A tutorial level was implemented for Shacetha. The level itself is very short with the main goal being to find a blue crystal and defeat a ghost enemy, an image of the tutorial level can be seen in Figure 11. The main idea behind the tutorial level is to show players how the game is controlled and some of the gameplay mechanics. In Shacetha this is done through use of text boxes with information about the controls and short descriptions about certain aspects like the inventory. Another important part of the tutorial is to show players some of the gameplay to allow them to get familiar with it and to show off some of the enemies they will encounter in the game, which includes both the basic enemies and the bosses. After defeating the first boss in the tutorial the player is also shown the reward screen allowing them to get familiar with it. From playtesting the tutorial has been successful in introducing the game to players and didn't go through many changes from its initial design, but was limited to only one level with two separate parts i.e. an outside area with a slime enemy and an area inside a mountain with the ghost boss and skeleton enemies.

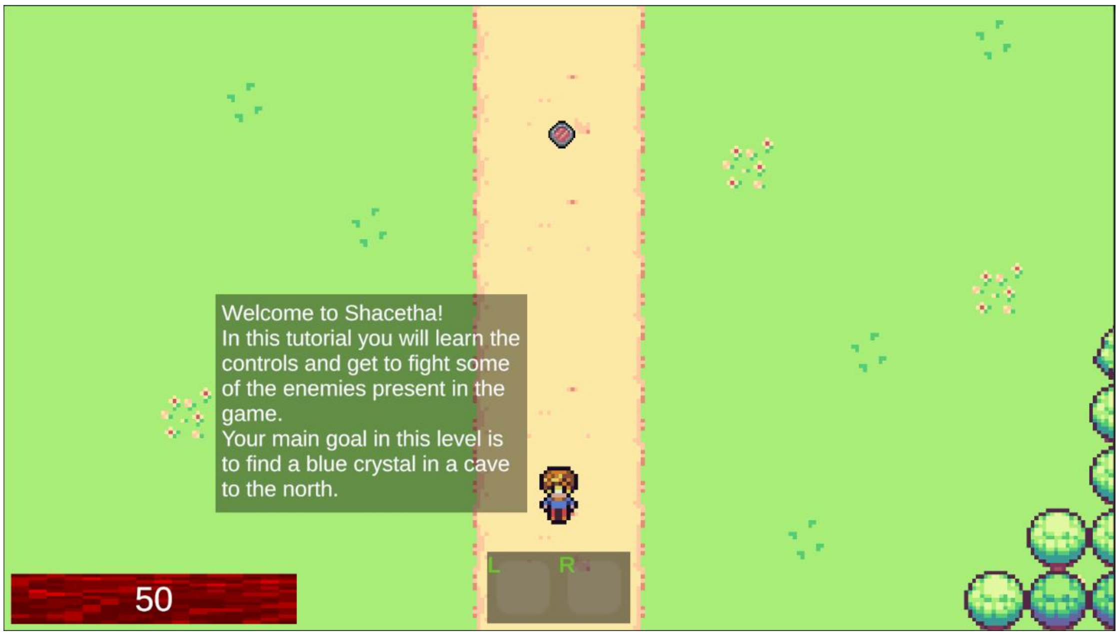

Welcome to Shacetha!
In this tutorial you will learn the controls and get to fight some of the enemies present in the game.
Your main goal in this level is to find a blue crystal in a cave to the north.

Figure 11. Start of the Tutorial Level

4.4 CHARACTERS

4.4.1 Changes to Character's Stats

The main structure of characters remained as planned in the design stage, more information can be found in section 2.2.1, but it has gone through some small changes. Character's range stat is no longer used to calculate the equipped weapon's reach. The range stat is used by enemies to calculate if their target is in their search range and attack range in their BT. All weapon's range are based on their animation i.e. a sword swings up and down having a constant range while a spear thrusts forward to reach enemies that are further away. Also the armour stat had to be rebalanced because of the new armour formula discussed earlier in section 4.3.1.

4.4.2 Difficulty and Enemy Variants

From playtesting the arena mode, it quickly became clear that there was a need for some sort of difficulty increase as the player reaches higher levels. The initial implementation increased the number of enemies created based on the current level, but this quickly became overwhelming with the amount of enemies present. The current system contains different variants for same type of enemy i.e. a basic chase skeleton enemy has three difficulty variants easy, medium and hard and then within those difficulties it also has some other variants such as having different weapons equipped. This adds more variety to the enemies found in Shacetha. These harder variants have increased stats, and usually differ by their sprites and the items they have equipped. This allows for different class of enemies i.e. same enemy behaviour that the player is familiar with but slight changes to make them harder (Lambottin 2012). Most harder enemies have some change to them to make it clear to the player that they are facing a more difficult enemy, for example the harder enemies are larger in size and often have a different colour on their sprite such as the bat being green on medium difficulty and red on hard.

There are five different enemy types with three difficulty variants resulting in five different enemy scripts and fourteen different stat objects for enemies. Goblins don't have an easy difficulty variant. For bosses there are two different types with three difficulty variants resulting in six different stat objects for the bosses. Some of the variants differ by other things than just their stats for example the slime boss splits more times in the hard difficulty variant. All of the enemies and their different variants can be seen in Appendix A.

All enemies have tells to ensure the player is aware of an incoming attack and can learn these tells to be better prepared for these enemies in the future. The tells are visual, often having the enemy blinking a certain colour before an attack or ability. These have been effective in Shacetha to provide players with a way for players to learn the enemy behaviour patterns and could be further expanded to use the AudioManager to play a sound before an attack to give even more feedback to the player.

4.4.3 Summary of Enemy Types

The five different enemy types are as follows the ChaseEnemy, Runner, Bat, Spider and the Goblin.

ChaseEnemy

The ChaseEnemy is a simple enemy type that chases the player and uses their weapon as soon as they reach them. Their stats are average and are meant to be an easy enemy for the players to deal with.

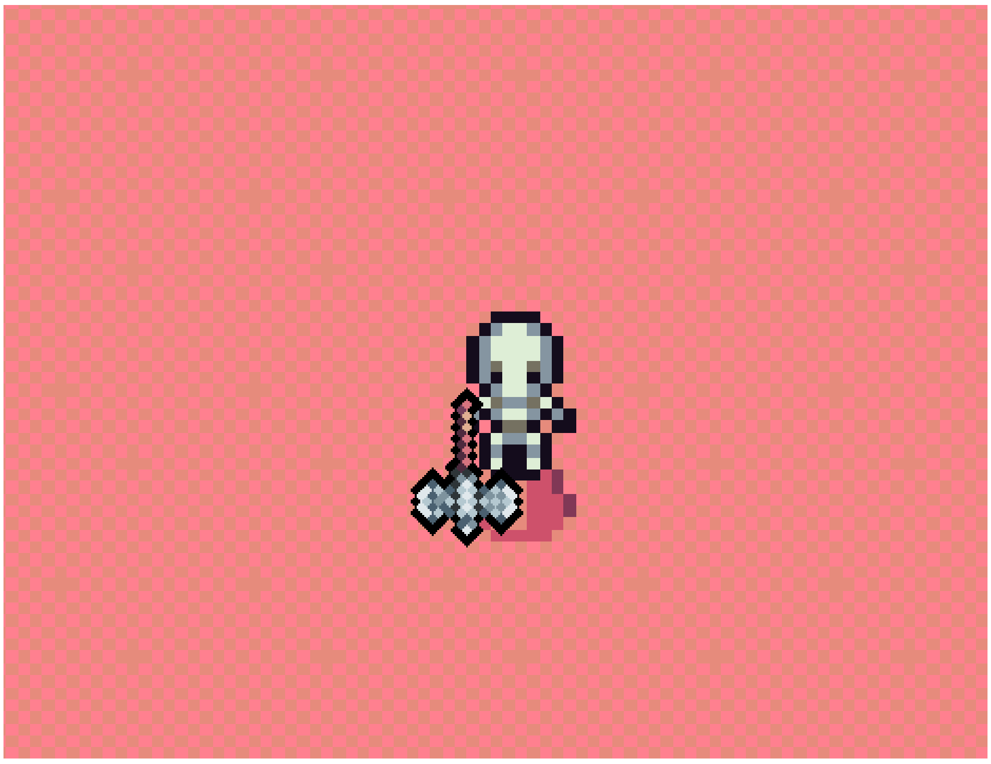


Figure 12. Chase Enemy

Runner

Runners quickly move towards the player and swing their weapons at them, after their attack they run away until ready to attack again. This enemy can deal large amount of damage and can be difficult to hit while running away, but once the player catches them, they are easy to defeat. This enemy type is meant to keep the player alert and moving as they will need to dodge the attack and then chase the enemy.

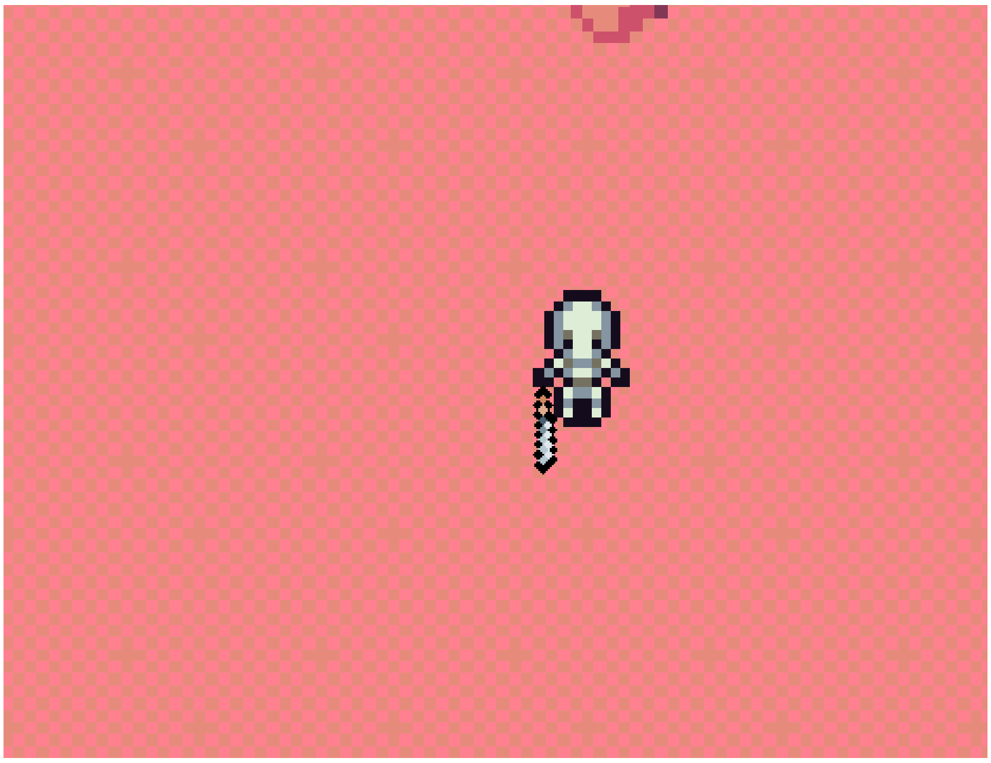


Figure 13. Runner Enemy

Bat

The Bat enemy had two versions. In the first iteration, this enemy would update the player's position once every five seconds, this tried to simulate bat's echolocation. This iteration resulted in an enemy that would usually have very unpredictable movement that goes against player's ability to learn enemies and often resulted in them not being an actual threat to the player as their movement often avoided the player. In the second version the Bat now updates player's position every time player uses an item. Showing a blood icon above the enemy once they update the position as a tell for the player. This results in interesting behaviour for which the player must plan around, as if they are close to a Bat and use an item they will be in great danger. In both versions the Bat moves in a zigzag motion, initially it used the Sine function to achieve this but resulted in bugs and as such was reworked to use the Bat's invisible hands to move between them. The resulting behaviour provides a player with a constant threat that is interesting and requires strategy and planning to defeat.

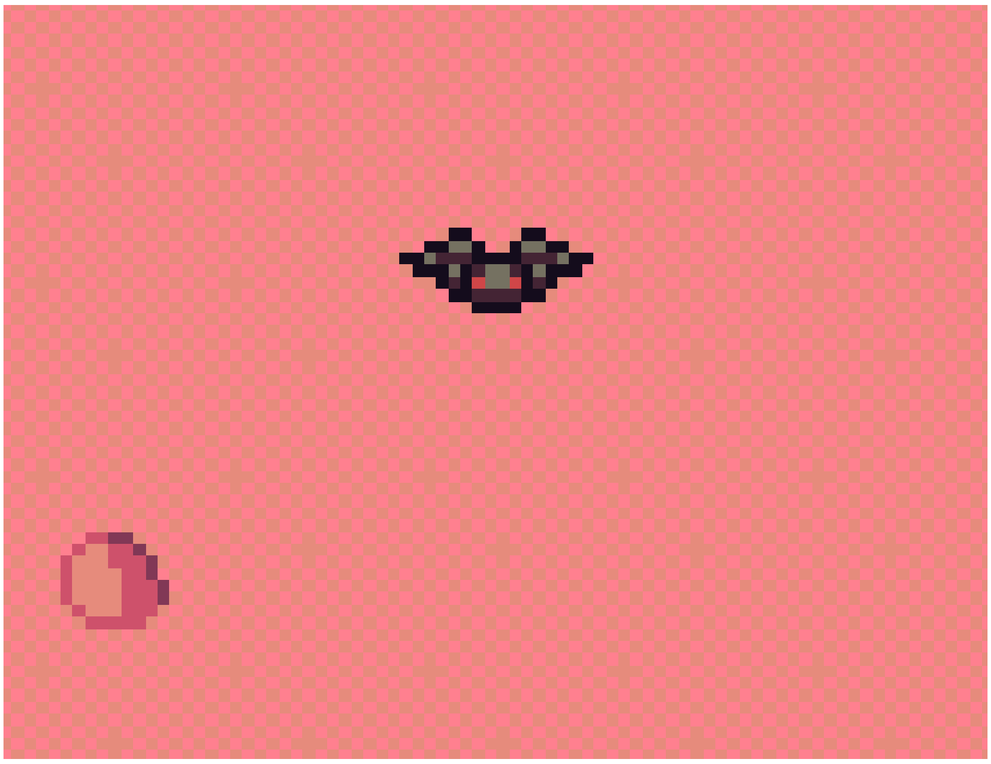


Figure 14. Bat Enemy

Spider

Spiders in Shacetha are very agile enemies that need to get close to the player to deal massive damage with their fangs. Spiders have the ability to create a web that speeds it up and slows all other characters. This ability is used after performing certain amount of attacks, initially set to every three attacks but is different depending on the difficulty variant. This type of enemy is very dangerous and is usually one of the first targets the player should focus on, they can quickly defeat the player and their slowing web can cause player to be caught in a bad position and get surrounded by their opponents but other enemies also get slowed by the web which the player can use to their advantage to escape danger.

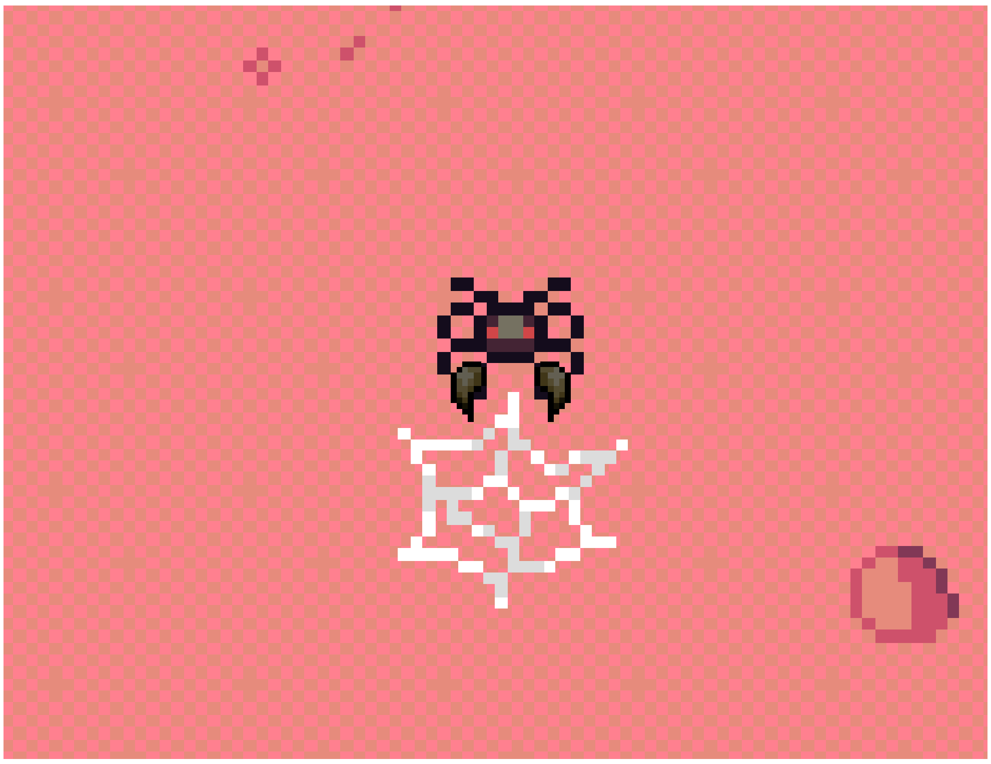


Figure 15. Spider Enemy

Goblin

The last type of an enemy is the Goblin, they are very quick and tend to be more problematic for players. They have an ability to throw rocks at their target and will run away if the player starts getting too close. Unlike the Runner which is usually vulnerable while running away as their weapon is on cooldown, Goblins will swing back if the player does get close enough. The equipped daggers can deal huge amounts of damage and should be avoided. From playtests this enemy has been found to be more difficult compared to other types, and a constant threat as it throws rocks at the player. Due to this difficulty the Goblin only has medium and hard difficulty variants to prevent them from appearing in the early levels.

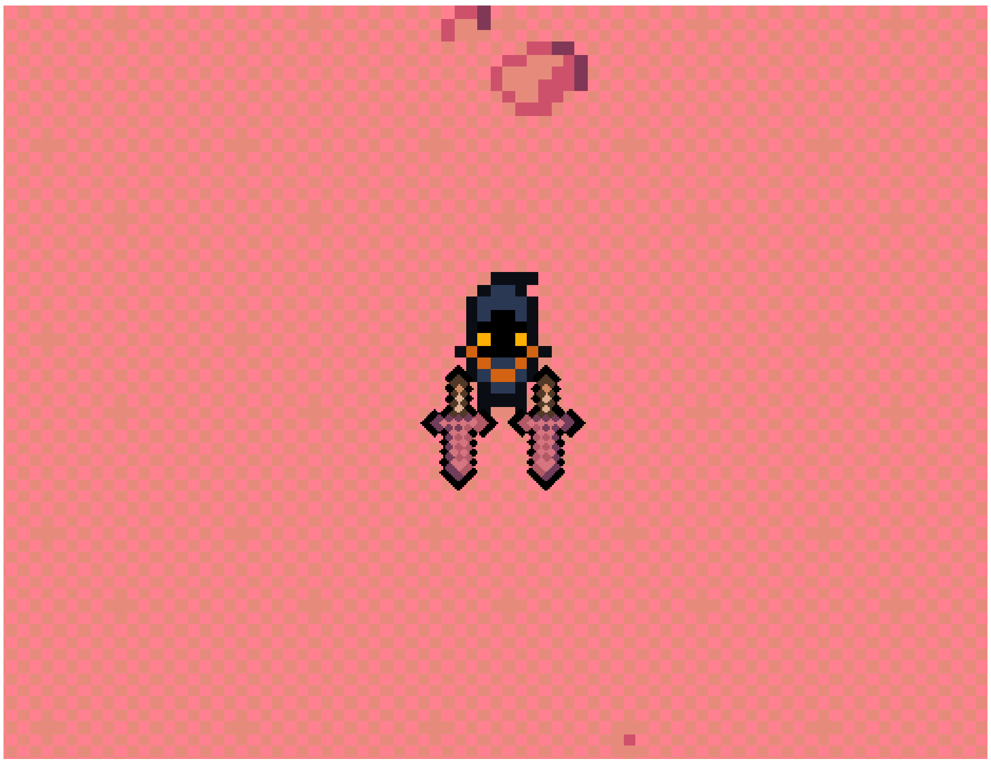


Figure 16. Goblin Enemy

The two different boss types are the Dasher and Slime.

Dasher

The Dasher is a ghost that has large max HP, good amount of amour, deals deadly damage but is slow. As the name suggests, Dasher has a unique ability to dash at the player, turning to an orange colour as its preparing to use it. Player must react quickly to dodge this ability and should also try to keep at a distance to ensure they have enough room to avoid the attack.

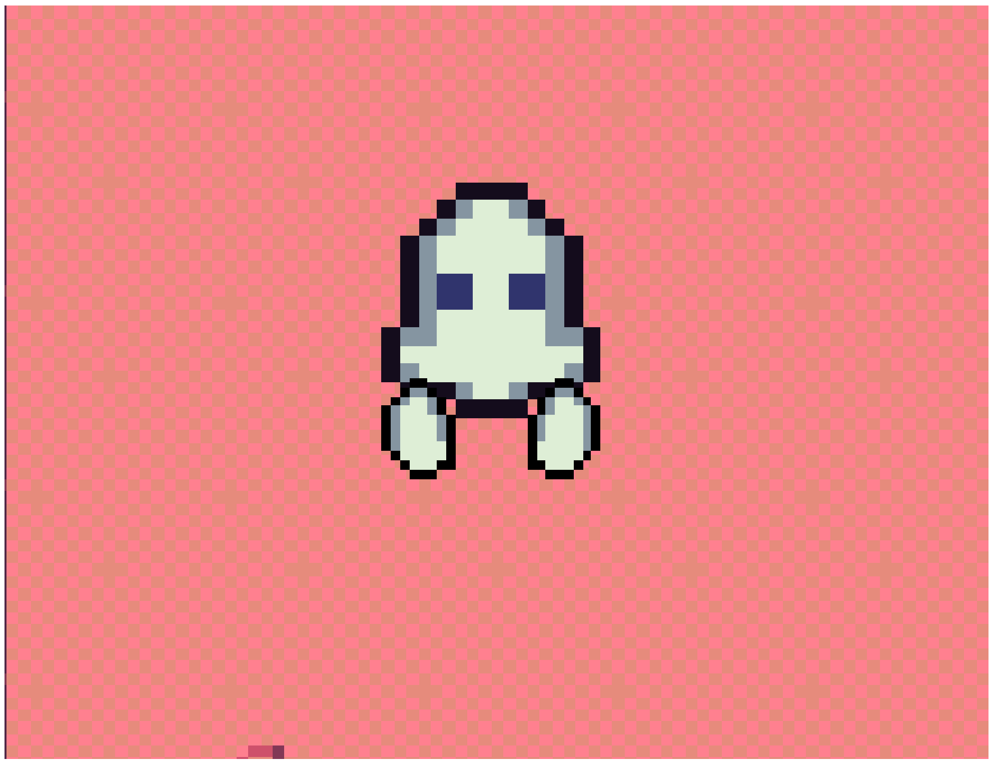


Figure 17. Dasher Boss

Slime

The Slime boss can only move by jumping, during the jump it cannot be damaged and cannot deal damage but will deal massive damage if it gets in contact with the player after it lands. This boss has several special abilities, firstly it spawns a poison puddle when it lands after a jump, this will deal damage to the player if they happen to walk over it. The Second ability is that the Slime will explode when it is close to dying, dealing damage around it and splitting into two smaller versions of itself. There is a max amount of times a Slime can split which is pre-defined in the enemies' script. For easy and medium difficulty, the Slime can split two times and for the hard variant it splits an additional time for a total of three times.

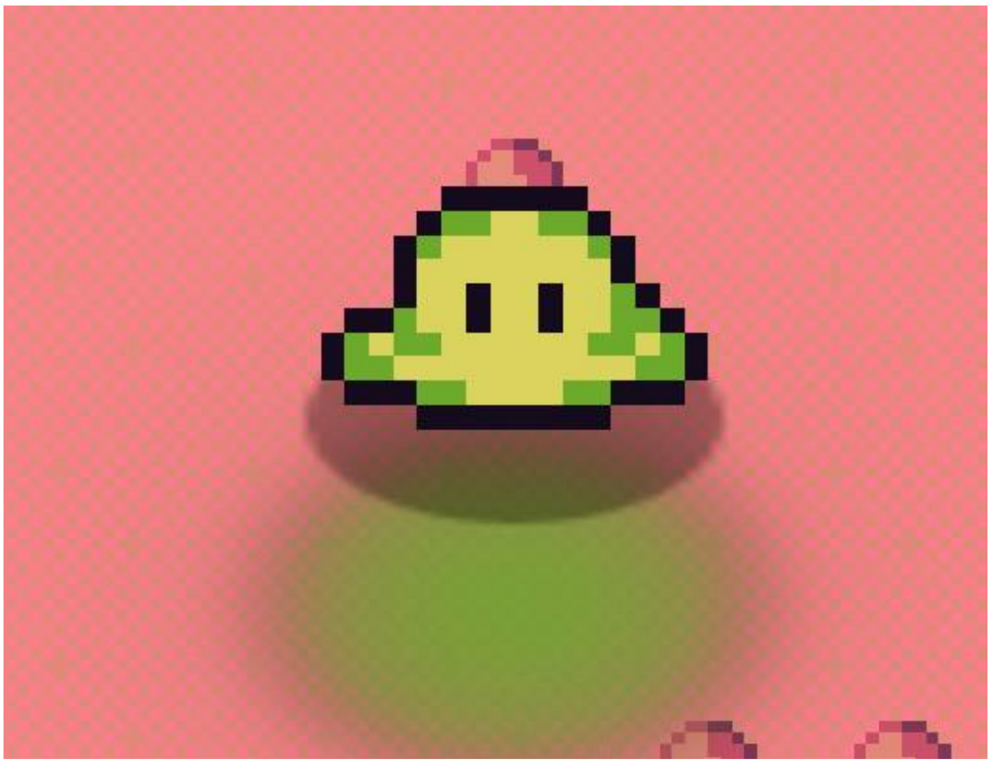


Figure 18. Slime Boss

4.4.4 Changes in BT Design

The implementation of the BTs followed the initial design found in section 2.3.5 very closely, all enemy types have their own BT that is created as soon as the enemy gets initialised in the game world. The main difference with the final BT design is that some of the nodes were changed and new nodes were added to help produce interesting enemy behaviours. These include a *CheckBoolNode* which main use is to check if an ability is on cooldown. This node can be used to check other conditions for example in the case of the spider enemy it checks if the spider has done certain amount of attacks and will use their ability if that is the case, more details in section 4.4.3.

The search and attack range nodes were simplified to a single *RangeNode* in which the range value can be set. This allows to use the *RangeNode* for situations such as to check if the target is in ability range, attack range or runaway range and then execute desired behaviour after, using corresponding nodes for example *AbilityNode*, *AttackNode* or *RunAwayNode*. The *RunAwayNode* is a new node that helps to separate an enemy's logic for moving towards their target and running away from their target, this helps create enemies like the goblin who can often dodge players attacks by running away.

The *Repeat* node was removed as currently there was no need for such node, and similarly the probability node was not implemented as there is currently no enemy that would utilise it. The summary of the final BT nodes can be seen in Table 4, and all the enemy BTs made for Shacetha can be found in Appendix B.

Node Name	Node Type	Behaviour
Node	Base	Basic abstract node that all other nodes inherit from
Selector	Control	Executes nodes until the first success or all failed
Sequence	Control	Executes nodes until the first failure or all success
AbilityNode	Leaf	Returns success after executing the ability method
AttackNode	Leaf	Returns success after executing the attack method
ChaseNode	Leaf	Returns success after executing the move command or if it has reached its destination
CheckBoolNode	Leaf	Returns success if the checked Boolean is true otherwise returns failure. Main use is to check if an ability is on cooldown
IdleNode	Leaf	Returns success after executing the idle method
RangeNode	Leaf	Returns success if the target is within the specified range
RunAwayNode	Leaf	Returns success after executing the move away method
Inverter	Decorator	Returns opposite result of its child node

Table 4. Summary of the Final BT Nodes

4.5 POLISHING

4.5.1 Fixing bugs

During playtesting there was a number of bugs found, any bug that prevented further implementation was fixed as soon as possible while smaller bugs were documented and added to Trello board or the GitHub repository, these included issues like wrong animation being used, wrong sounds played or incorrect values used. It was important to write the required information about the bug to be able to find them and/or reproduce them at a later stage. After all the main parts of the implementation were finished the bugs were fixed one by one.

4.5.2 Final Balancing

As mentioned in sections 3 and 3.2, it is very important to keep games balanced to ensure the game provides enough of a challenge while being fair to the player which leads to an overall better enjoyment of the game.

For Shacetha the game went through many balancing changes, for example initially the player started with 100 max HP, then was changed to 70 and finally ended with a value of 50. A lot of the values used for stats were picked from looking at other similar games or picked randomly between ranges based on other characters that were already added, this was a good starting point to start play testing.

The stats of all items and characters have been added to excel tables, this can been seen in Table 6 and Table 7 found in Appendix C. This allows to easily view all the values and fix any outlier values, for example if all enemies had 50HP and one had 150HP it could indicate that there is some unbalance and required further investigation. It is important to remember that some values can differ by large amounts, some enemies have higher armour values, also higher difficulty variants of enemies will have overall higher stat values. After play testing it was important to write down which parts were too difficult or too easy. This information was then used to change the values in the excel tables and finally in the Unity editor the changes were applied by adjusting the scriptable objects responsible for holding the item’s or character’s stats. This process was then repeated until the new values had the desired effect.

This approach was very effective in quickly catching things that stand out a lot such as one enemy doing huge damage and killing the player in one hit, and was important part to keep Shacetha challenging enough without being impossible for players to reach higher levels.

5 EVALUATION

5.1 REFLECTION ON PROJECT OBJECTIVES

Undertaking a project like Shacetha proved a great way to learn different game design techniques and some useful patterns that are often used when creating games but can also be used in other types of software. This includes the singleton pattern which can be seen used in many games and is not limited to games made using Unity. For Unity specifically the use of scriptable objects is a great way to ease some aspects of creating a game for example creating stats for items and enemies became very simple with scriptable objects and was a great way to quickly make balance changes. Scriptable objects are also very useful when creating different systems for the game such as the event system discussed in section 4.1.2, this is a very effective way to add more complexity to a game without requiring huge amounts of code.

Another great benefit from this project came in learning what goes into creating clear and useful UI for the user. While the UI in Shacetha still has areas to improve, it is clear and effective at providing information to the user without taking the focus from the game itself. More information on the design of UI in Shacetha can be found in section 2.4.

5.2 SUCCESSES

The final combat system and enemies implemented in Shacetha has achieved the desired effect of allowing players to have challenging fights that they can learn thanks to the enemy tells before attacks. From playtesting it was clear that at first the game can be difficult with players not reaching very high levels in the endless mode, but after playing the game enemy attack patterns became more clear, players learned how avoid a lot of them and resulted in reaching higher levels. This is exactly what Shacetha aimed to achieve, also having the highest level achieved displayed in the main menu was a great way to encourage some competition between players. Having this learning and competitive aspect to the game creates an engaging experience that players want to come back to (Koster 2013).

Shacetha reached its goals by having different systems that work together effectively. The enemy BTs create interesting behaviour, all NPCs use Unity’s animation system to provide tells and carry out their actions for example attacking, and the AudioManager adds sounds to provide audible feedback to the player and it also makes actions have more impact as the player can hear their sword swinging and when enemies take damage (Sabbagh 2015).

5.3 Issues and Failures

The tooltip system is very important tool to provide players with information about items and can also be extended to work with other parts of the game if required. The main issue with this system is that in its current state it has proved to be prone to causing bugs. A lot of the final bug fixing done for Shacetha involved the tooltip system, often the issues were with tooltip going offscreen, displaying incorrect information or not closing properly when player closed part of the UI for example the inventory window. The bugs were fixed but there is situations where the tooltips still don’t behave as intended, and some of the fixes would not work well if the game got expanded in the future, as such this is one area that would require good amount of time to rework.

The reward system in Shacetha works as intended and provides players with useful rewards as they defeat more levels of enemies. It can also often offer interesting choices to the player by having them choose between healing but potentially missing out on a powerful item, giving them the control over how they want to play the game. This all works very well in the current form of the game, but the main drawback of the current reward system is when it comes to scalability. Currently when new items are added the item lists increase in size making it less likely for players to receive an item that will fit their playstyle. This also has an issue of making it hard to design items that work at all points of the game i.e. currently items are available as potential reward at all times which could mean that a lucky player could get an epic rarity item that would deal massive amounts of damage for the part of the game they are at. The stat reward in the current system also becomes less useful as player reaches higher levels as the stat values are pre-defined and don’t change. Ways to address these issues can be found in section 6.

5.4 COMMERCIAL VIABILITY

Success of a game is often based on their sale numbers, for Shacetha while the combat system does feel satisfying and the endless arena mode has been implanted, there is currently no plans to release it on any store. Compared to similar games in same genres it is still missing a lot features that would justify asking money for it. Games like Moonlighter and Undermine have similar combat systems but have other systems that the player must focus on i.e. in the case of Undermine it is reach the final boss and unlock more items and for Moonlighter it is to collect resources to upgrade a town to increase player's power for future fights and defeat the final boss. Shacetha currently focuses on the endless arena mode and has no final goal that the player can work towards. Features discussed in section 6 would contribute a good amount of content for the game, and after which it could be justified to list Shacetha for sale.

6 Future Additions

6.1 Expand the Tutorial Into a Story Mode

The story aspect of the game was out of scope for this project. The tutorial from section 4.3.3 ended up being a single level that introduces the game to the player, with very limited story elements i.e. limited to telling a player they need to find a blue crystal in the mountain. If the project were to be continued in the future, adding a fully structured story mode with multiple levels, characters and goals players can aim to complete would allow the game to appeal to a larger audience, as there is a big player base that enjoys narrative games. It would also provide elements that would be introduced for the story mode but could be re-used in the arena mode for example enemies, bosses, special items etc.

6.2 Expand the Combat System

Current combat system has achieved its goals and feels satisfying but if given more time there are areas that could be improved and expanded. First adding special stats to items would make for more interesting rewards and more complex combat system. Status effects like poisons, burning, shocked etc. are great ways of adding special effects to items and enemies that the player must watch out for or can aim to use themselves to defeat harder enemies. This would also allow to experiment with ways to make certain elements strong against other ones which could add more strategy to how a player picks their items and would require them hold onto some items because of an effect that item might have. This type of addition would take a lot of prototyping to ensure the new additions fit well with the rest of the game and that none of the effects become to unbalanced. Overall, this would be a great addition to game like Shacetha, providing another layer of complexity to it that should be enjoyable for all players.

One issue with combat system currently is that dual wielding weapons i.e. having a weapon in each hand, doesn't provide a huge benefit. Enemies have invincibility frames which prevent them from taking damage for some time after getting hit. This means that having two weapons equipped usually is only good to be able to quickly use two different weapon types, but same could be achieved by switching weapons mid combat. Adding some form of bonus for dual wielding for example extra damage or special attack, would make it a more practical option and would allow more unique playstyles for players to choose from. Some examples of a special attack could be using two spears allows players to throw one of them or two axes allow characters to use a whirlwind ability which would spin the character and deal damage to all enemies around the player.

6.3 IMPROVE THE REWARD SYSTEM

Adding items with special effects would make the rewards more interesting, but the main focus for the reward system would be to ensure the rewards feel powerful enough as the player reaches higher levels. Currently the item rewards are completely managed by the RewardManager, but one improvement to the system would be to use a separate script that would handle item lists. Such a system was started in Shacetha and can be seen in the *so_ItemDatabase.cs* script, this would allow the database to handle keeping track of all the items present in the game, and to add logic for distributing these items based on power level and rarity. It became quickly apparent that this system would take a lot of time to get working and tuned to provide satisfying rewards to the player, as such it was out of scope in the context of the final year project. Finishing this system would provide great extensibility and scalability to the game as new items could easily be created and added to the database, and creating special powerful items that have different unique elements such as the status effects, could be handled by the item database script. The reward manger would then be able to request items from it and receiving items that scale based on the current level.

Another important part that would need to be worked upon would be the stat reward, this reward currently has pre-defined values that don't increase with levels and difficulty. By having the item database handle the items, RewardManager could be expanded to scale the stat reward based on the current level to keep it a viable option through all stages of the game.

6.4 General Improvements

This section covers any other additions that would be important if Shacetha was planned to be released for sale, these focus on adding more content to the game, some quality of life improvements e.g. save files, and code changes to improve performance as the game would grow in size.

First for performance the use of object pool pattern would be added to the ArenaManager for producing enemies. The object pool pattern is often used in video games and works by keeping a pool or list of objects, instead of instantiating a new object every time the ArenaManager needs to create an enemy it would be able to use one of the already existing objects found in the object pool. Once the object is used i.e. an enemy is defeated, it would go back to the pool to be available for future use. The object pool would only need to instantiate new enemies once all the already existing objects were already in use. This improves performance thanks to reducing instantiation time, memory allocation and freeing. Finally it also reduces problem of memory fragmentation which allows the object data to be stored in one contiguous block rather than many smaller blocks (Wang 2016).

Adding new items and enemies would be very important to increase the variety in the game. It is also a great way to make new and interesting encounters for players to defeat, adding to the replay-ability of the game. New arenas are another area that could be added to for the same reasons. In relation to these items and enemies, expanding the score system should be looked at. Currently Shacetha only keeps track of how high of a level a player reached, by having a more complex scoring system it would add to the competitive aspect of the game. The system could keep track of the enemies defeated, reward points for not taking damage during a level, which would provide players with more challenges that they could aim to achieve. Players could challenge each other, for example highest amount of levels with no-damage taken or defeat 'x' amount of boss levels etc. An online leader board would also be planned to be added to the game. This would be a great way to save players achievements such as highest level reached and would allow players to view other people's stats and compete with each other.

Another system that would be added to Shacetha would be a save system. A save system would be a great way to allow players to take a break from the game when required and come back at a later stage. Currently Shacetha uses Unity's PlayerPrefs (https://docs.unity3d.com/ScriptReference/PlayerPrefs.html) class to save the highest level achieved by the player, this system allows for storing data on the user's machine and stores it in different areas depending on the user's operating system. This system would be altered to use a different saving method to save the players current state for example items in inventory, current level, stat values etc. and then allow to be loaded back again to start from where the player left. This type of system often uses JavaScript Object Notation (JSON) or Extensible Markup Language (XML) formats to store the data and Shacetha would most likely use JSON to save its game state but this would require further research to ensure the right techniques are used.

Appendix A Enemies in Shacetha

This appendix contains all the different enemies present in Shacetha, separated by their difficulty. For more information see sections 4.4.2 and 4.4.3.

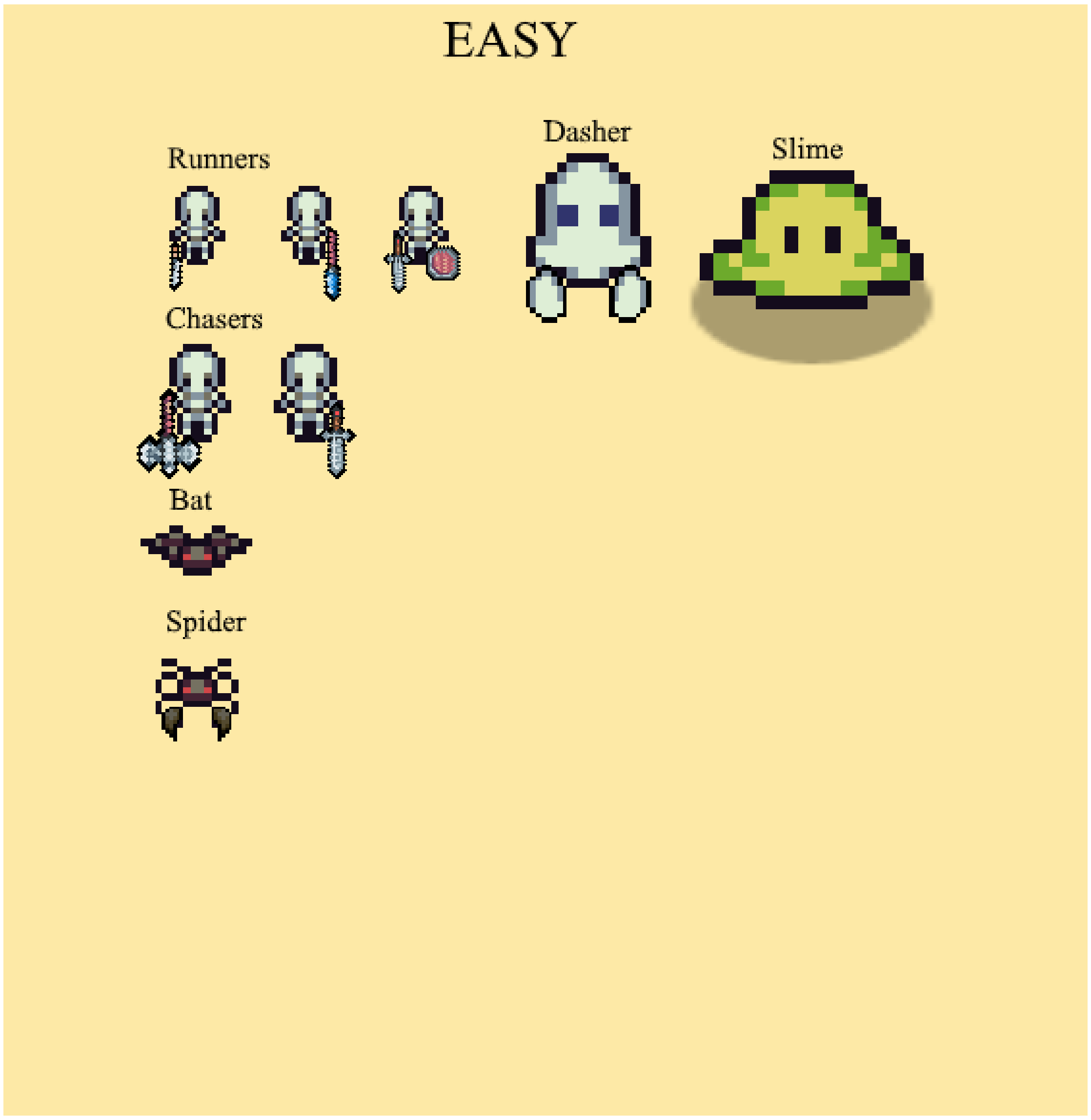


Figure 19. Image of the Easy Difficulty Enemies

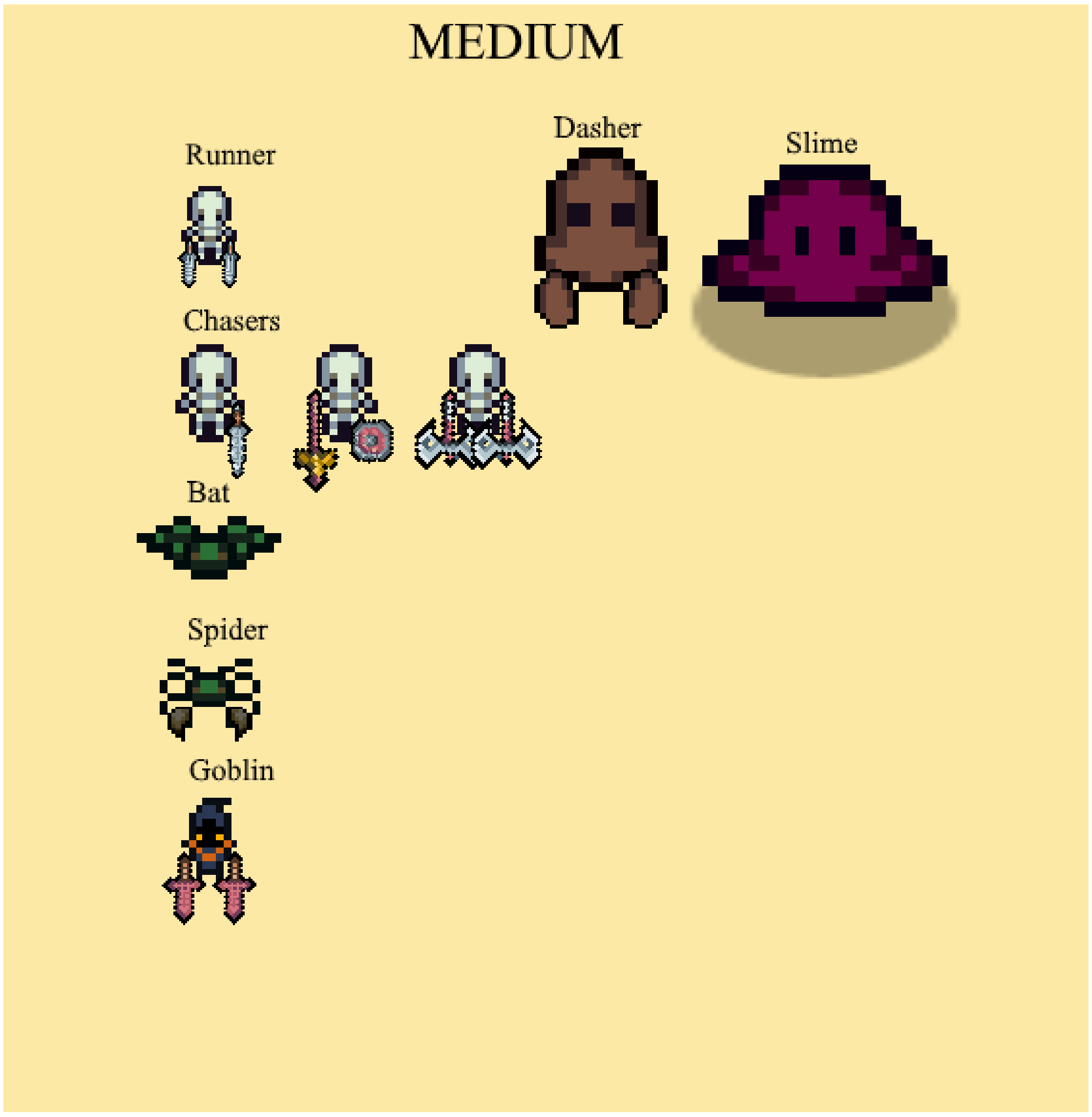


Figure 20. Image of the Medium Difficulty Enemies

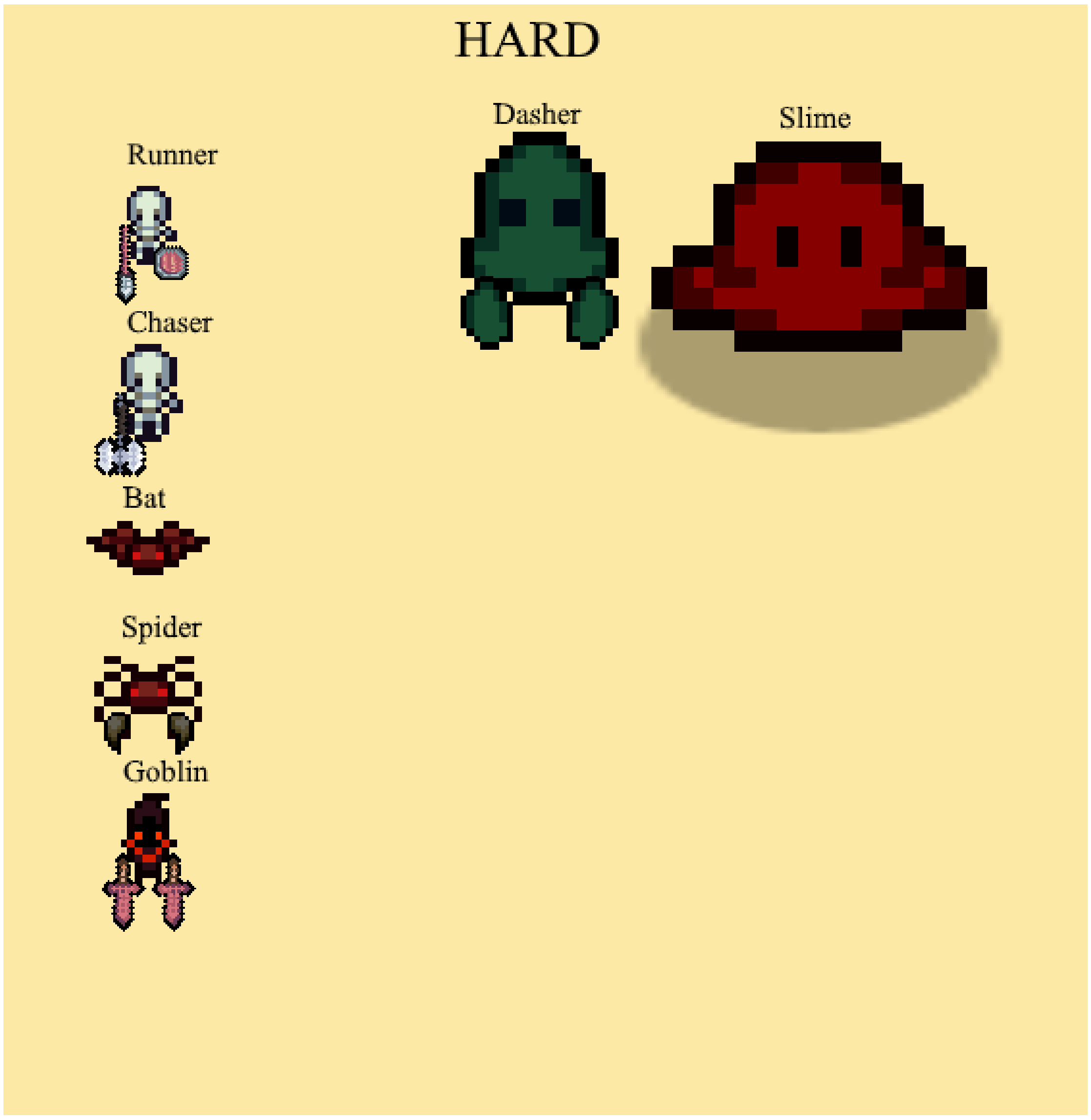


Figure 21. Image of the Hard Difficulty Enemies

Appendix B FINAL ENEMY NPC BTS

This appendix contains diagrams for all the NPC BTs. These BTs are responsible for how the enemies in Shacetha will behave. More information on how the BTs were used in Shacetha can be found in sections 2.3.5 and 4.4.4.

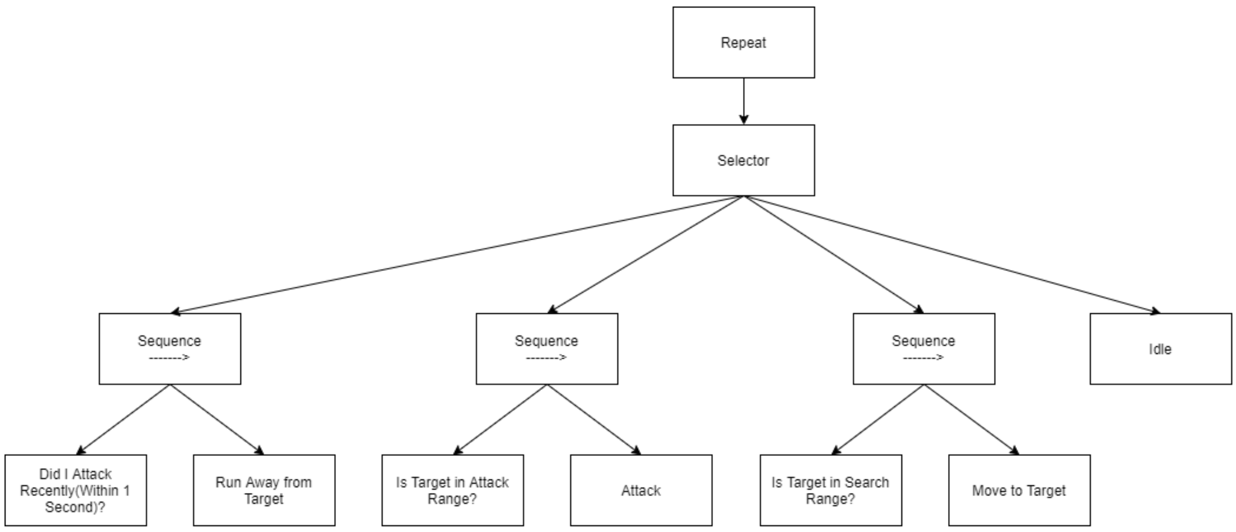


Figure 22. Runner BT

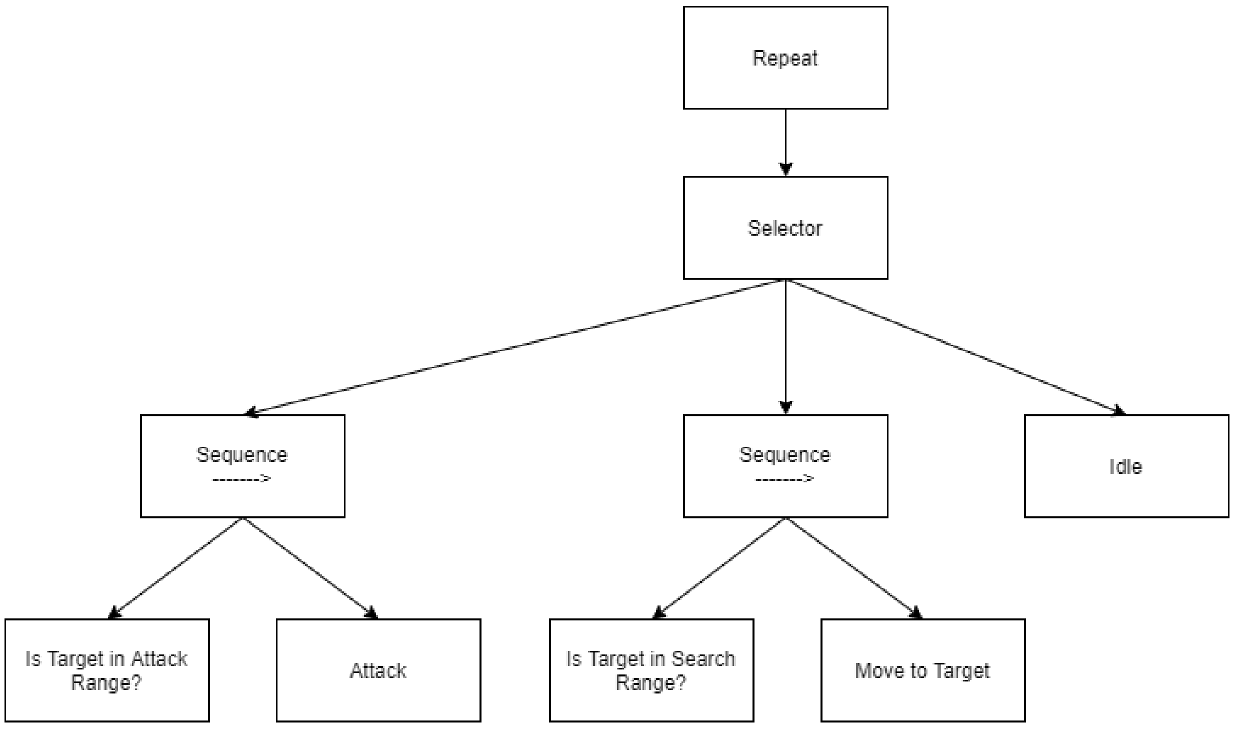


Figure 23. Chaser BT

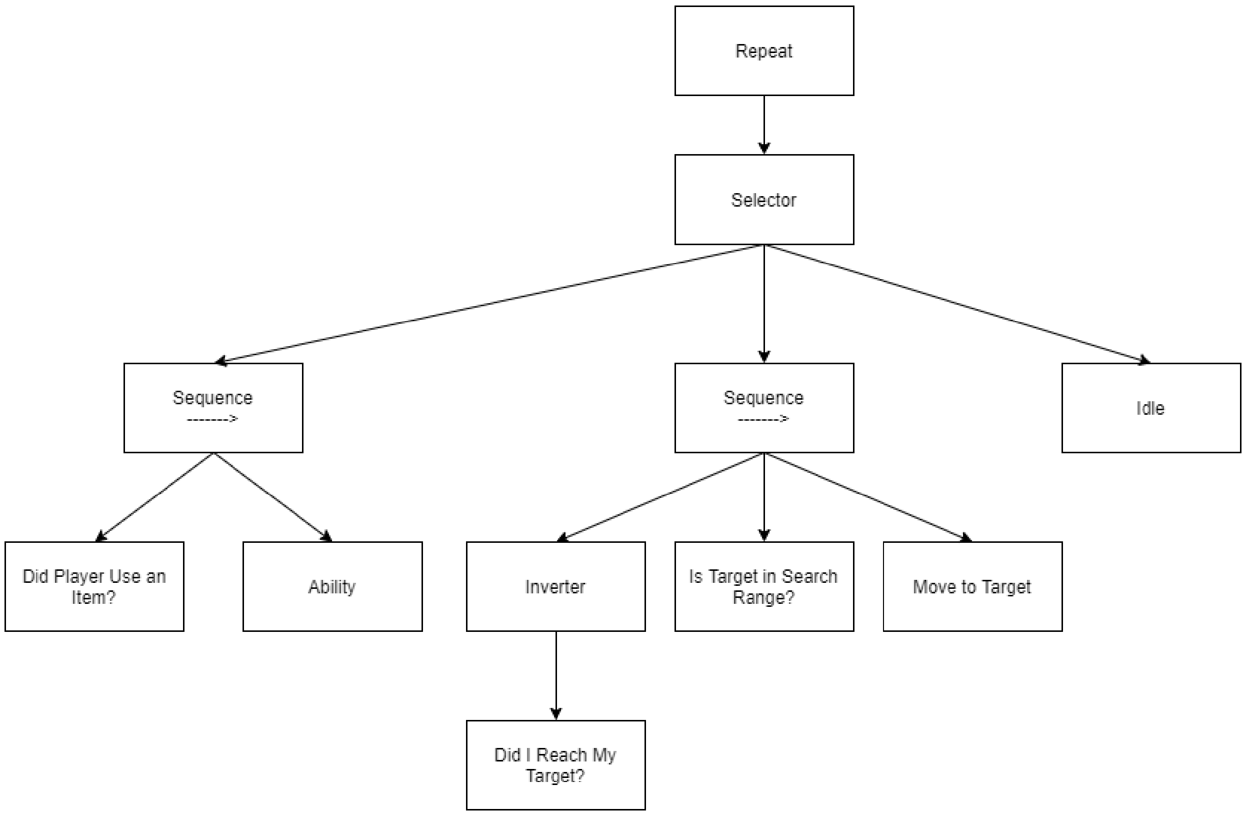


Figure 24. Bat BT

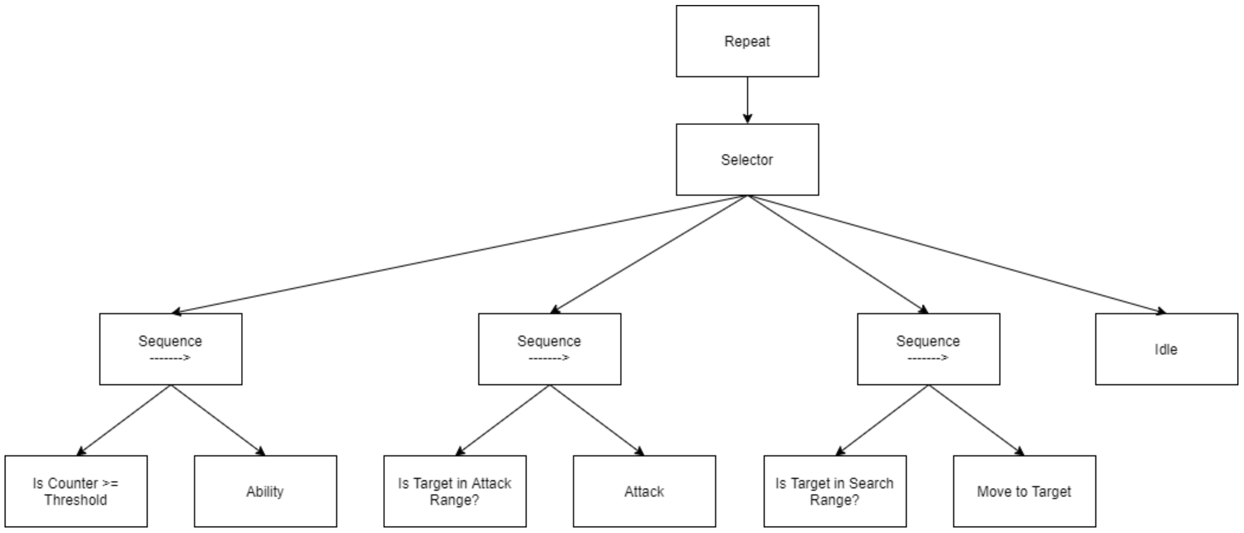


Figure 25. Spider BT

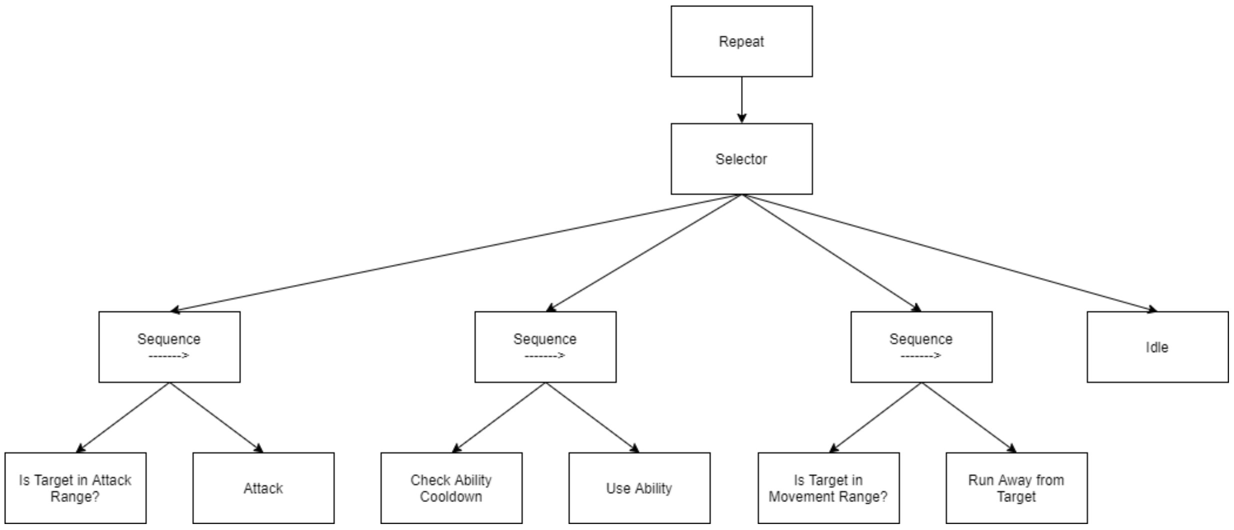


Figure 26. Goblin BT

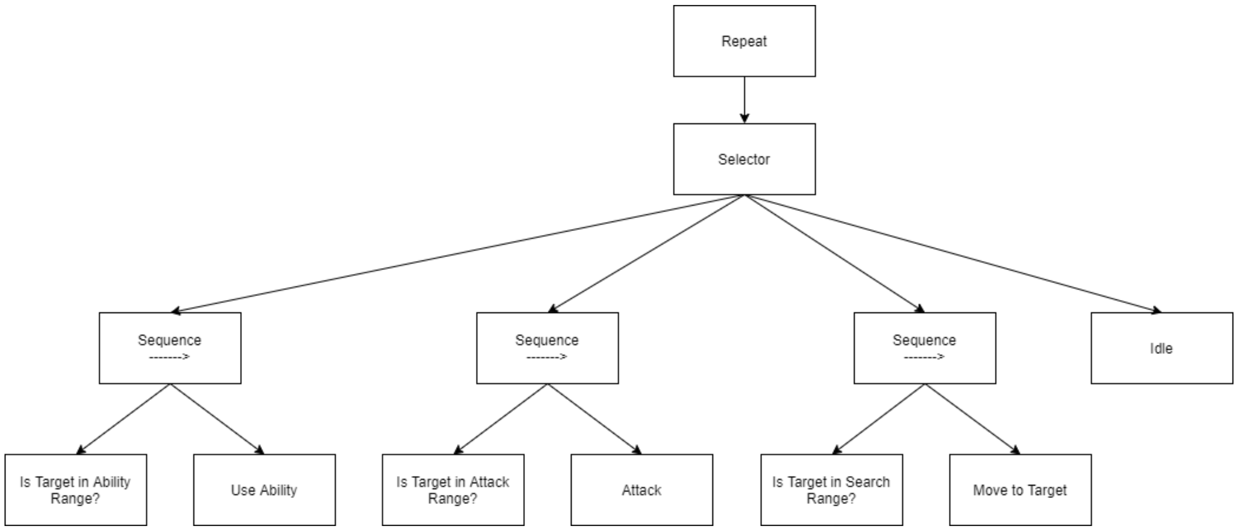


Figure 27. Dasher BT

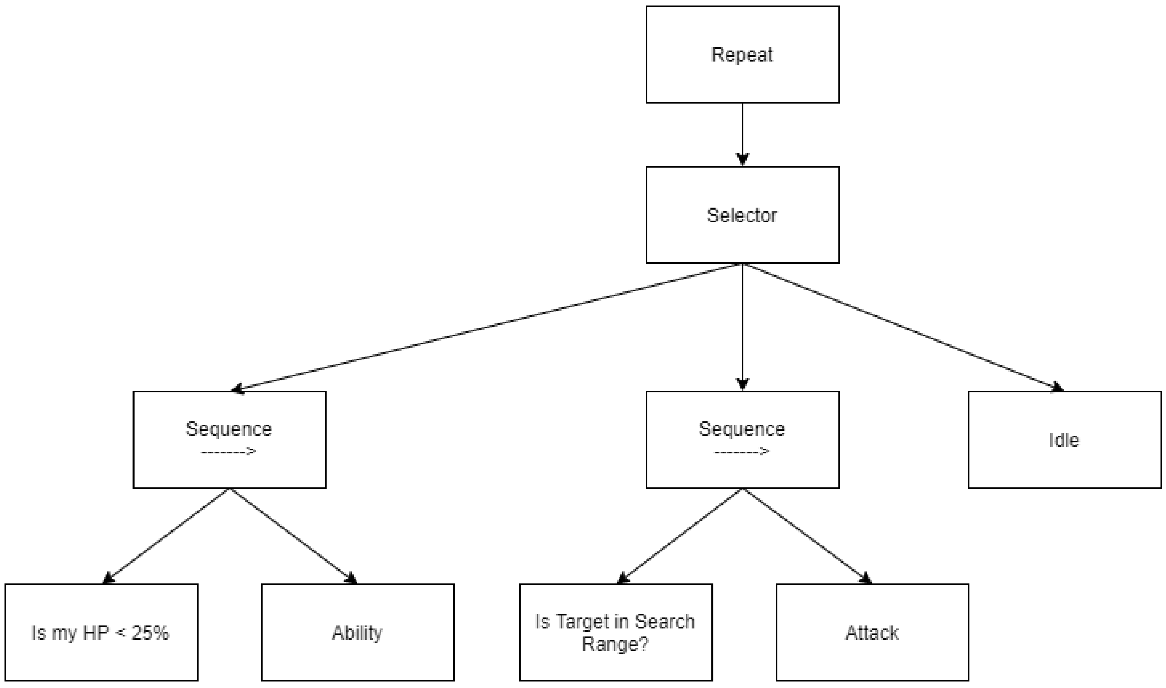
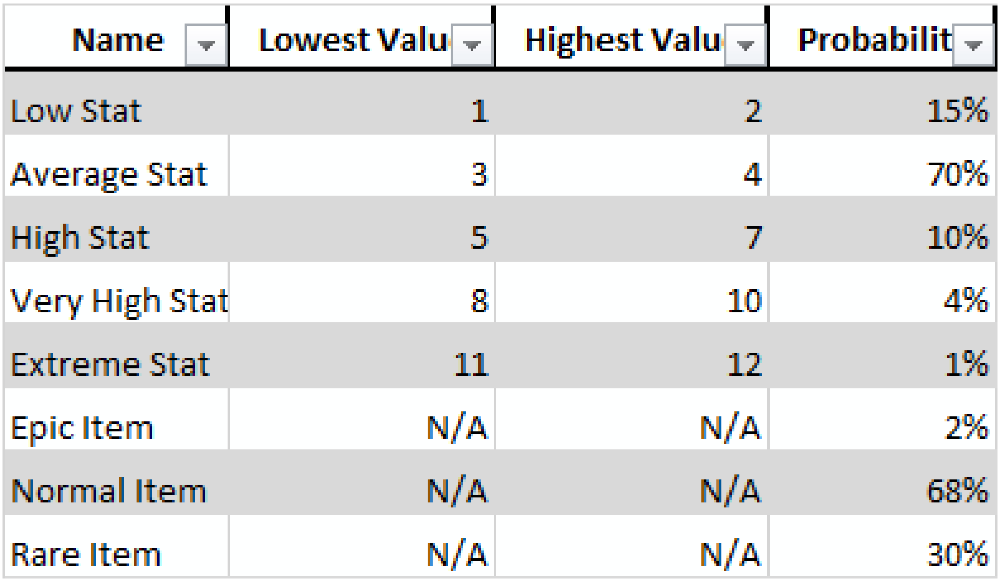


Figure 28. Slime BT

Appendix C STATS AND PROBABILITY TABLES

This appendix contains three tables made using Excel. First table shows the probabilities for the reward system. Second table contains the stat values for the different NPCs present in Shacetha. Last table contains the stat values for the different items. All these tables were used when balancing the game to view and compare the different values for the NPCs and items, allowing to change the values in the tables first and then doing the changes in the Unity editor. More information on the balancing process can be found in section 4.5.2.

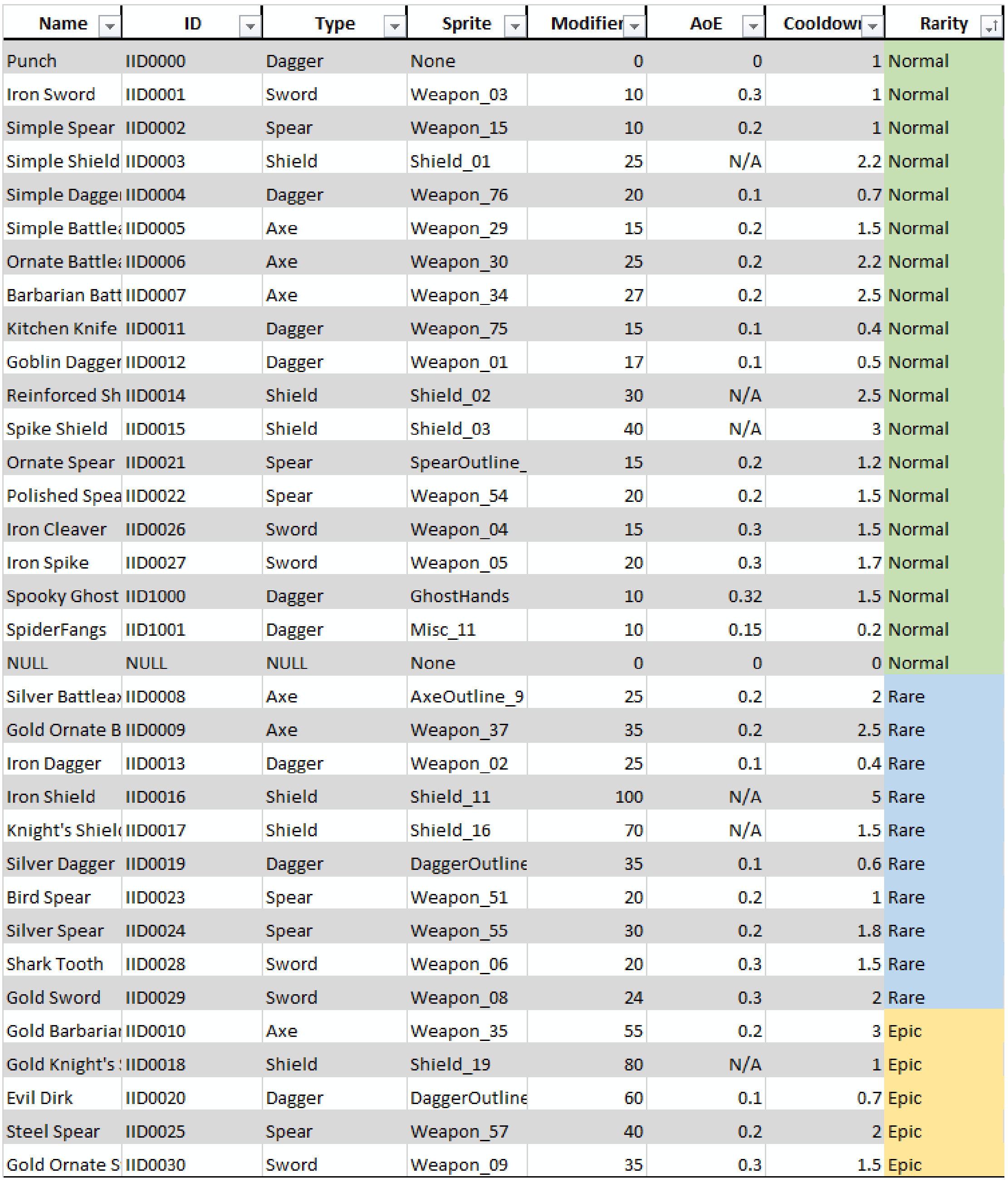


Name	Lowest Valu	Highest Valu	Probabilit
Low Stat	1	2	15%
Average Stat	3	4	70%
High Stat	5	7	10%
Very High Stat	8	10	4%
Extreme Stat	11	12	1%
Epic Item	N/A	N/A	2%
Normal Item	N/A	N/A	68%
Rare Item	N/A	N/A	30%

Table 5. Probabilities for the Reward System

Name	Type	Max HP	Damage	Armor	Speed	Range	Difficulty
Player	Player	50	5	0	4	0.5	N/A
TestEnemy	Basic Enemy	999999999	10	999999999	0	0	Easy
Runner	Basic Enemy	25	7	0	3.5	0.75	Easy
ChaseEnemy	Basic Enemy	40	5	10	2.5	1	Easy
Bat	Basic Enemy	30	15	0	2.8	0.5	Easy
Spider	Basic Enemy	30	10	0	3	1	Easy
Goblin	Basic Enemy	50	5	5	4	0.7	Medium
Runner	Basic Enemy	40	15	10	4	0.75	Medium
ChaseEnemy	Basic Enemy	60	10	20	3	1	Medium
Bat	Basic Enemy	50	25	5	3	0.5	Medium
Spider	Basic Enemy	45	15	5	3.3	1	Medium
Goblin	Basic Enemy	80	10	10	4.2	0.7	Hard
Runner	Basic Enemy	60	20	15	4.5	0.75	Hard
ChaseEnemy	Basic Enemy	100	25	50	3.7	1	Hard
Bat	Basic Enemy	70	40	25	3.2	0.5	Hard
Spider	Basic Enemy	60	30	20	3.7	1	Hard
Dasher	Boss Enemy	100	20	20	2	1.3	Easy
Jumpy Slime	Boss Enemy	80	25	10	15	1	Easy
Dasher	Boss Enemy	250	40	30	2.5	1.3	Medium
Jumpy Slime	Boss Enemy	130	30	20	17	1	Medium
Dasher	Boss Enemy	320	50	50	3	1.3	Hard
Jumpy Slime	Boss Enemy	170	35	25	15	1	Hard

Table 6. All the NPCs' Stats



Name	ID	Type	Sprite	Modifier	AoE	Cooldow	Rarity
Punch	IID0000	Dagger	None	0	0	1	Normal
Iron Sword	IID0001	Sword	Weapon_03	10	0.3	1	Normal
Simple Spear	IID0002	Spear	Weapon_15	10	0.2	1	Normal
Simple Shield	IID0003	Shield	Shield_01	25	N/A	2.2	Normal
Simple Dagge	IID0004	Dagger	Weapon_76	20	0.1	0.7	Normal
Simple Battle	IID0005	Axe	Weapon_29	15	0.2	1.5	Normal
Ornate Battle	IID0006	Axe	Weapon_30	25	0.2	2.2	Normal
Barbarian Batt	IID0007	Axe	Weapon_34	27	0.2	2.5	Normal
Kitchen Knife	IID0011	Dagger	Weapon_75	15	0.1	0.4	Normal
Goblin Dagger	IID0012	Dagger	Weapon_01	17	0.1	0.5	Normal
Reinforced Sh	IID0014	Shield	Shield_02	30	N/A	2.5	Normal
Spike Shield	IID0015	Shield	Shield_03	40	N/A	3	Normal
Ornate Spear	IID0021	Spear	SpearOutline_	15	0.2	1.2	Normal
Polished Spea	IID0022	Spear	Weapon_54	20	0.2	1.5	Normal
Iron Cleaver	IID0026	Sword	Weapon_04	15	0.3	1.5	Normal
Iron Spike	IID0027	Sword	Weapon_05	20	0.3	1.7	Normal
Spooky Ghost	IID1000	Dagger	GhostHands	10	0.32	1.5	Normal
SpiderFangs	IID1001	Dagger	Misc_11	10	0.15	0.2	Normal
NULL	NULL	NULL	None	0	0	0	Normal
Silver Battlea	IID0008	Axe	AxeOutline_9	25	0.2	2	Rare
Gold Ornate B	IID0009	Axe	Weapon_37	35	0.2	2.5	Rare
Iron Dagger	IID0013	Dagger	Weapon_02	25	0.1	0.4	Rare
Iron Shield	IID0016	Shield	Shield_11	100	N/A	5	Rare
Knight's Shiel	IID0017	Shield	Shield_16	70	N/A	1.5	Rare
Silver Dagger	IID0019	Dagger	DaggerOutline	35	0.1	0.6	Rare
Bird Spear	IID0023	Spear	Weapon_51	20	0.2	1	Rare
Silver Spear	IID0024	Spear	Weapon_55	30	0.2	1.8	Rare
Shark Tooth	IID0028	Sword	Weapon_06	20	0.3	1.5	Rare
Gold Sword	IID0029	Sword	Weapon_08	24	0.3	2	Rare
Gold Barbaria	IID0010	Axe	Weapon_35	55	0.2	3	Epic
Gold Knight's	IID0018	Shield	Shield_19	80	N/A	1	Epic
Evil Dirk	IID0020	Dagger	DaggerOutline	60	0.1	0.7	Epic
Steel Spear	IID0025	Spear	Weapon_57	40	0.2	2	Epic
Gold Ornate S	IID0030	Sword	Weapon_09	35	0.3	1.5	Epic

Table 7. All the Items' Stats

ASSETS

All assets i.e. sprites, music and sounds used in Shacetha have either been purchased, are free to use with an appropriate attribution or have been released under public domain.

Lanea Zimmermanm, Character & Enemy Sprites, https://opengameart.org/content/tiny-16-basic

Blodyavenger, Items, https://blodyavenger.itch.io/rpg-items-retro-pack

Kenney.nl, Particles, https://kenney.nl/assets/particle-pack

Elthen, SpiderWeb, https://www.patreon.com/elthen

Raou, Sprites for Tilemaps, https://raou.itch.io/dungeon-tileset-top-down-rpg

Raou, Sprites for Tilemaps, https://raou.itch.io/topdown-rpg-pixel-art-tileset

Aleksandr Makarov, Sprites for Tilemaps, https://iknowkingrabbit.itch.io/heroic-building-pack

Aleksandr Makarov, Item Sprites, https://iknowkingrabbit.itch.io/heroic-icon-pack

3xBlast, Boss Music, https://opengameart.org/content/16bit-boss-battle-loop

CodeManu, BGM_action(In game music), https://opengameart.org/content/8-bit-music-pack-loopable

Oddroom, Menu Music, https://opengameart.org/content/music-loop-variations

Little Robot Sound Factory, Fantasy Sound Library, Website: www.littlerobotsoundfactory.com, https://opengameart.org/content/fantasy-sound-effects-library

Michel Baradari, Monster Grunt, Pain & Death Sounds, https://opengameart.org/content/15-monster-gruntpaindeath-sounds

Tuomo Untinen, RPG Sound Pack - Sound package from Heroes of Hawks Haven By, https://opengameart.org/content/rpg-sound-package

Kenney.nl, RPG Sounds, https://opengameart.org/content/50-rpg-sound-effects

ArcadeParty, Zombie, Skeleton, Monster Voice Effects, https://opengameart.org/content/zombie-skeleton-monster-voice-effects

Ogrebane, Ghost Sound, https://opengameart.org/content/ghost

REFERENCES

Bevilacqua, F. (2013) *Finite-State Machines: Theory and Implementation* *https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867*, available: [accessed 11 Jan 2021].

Bond, J.G. (2017) *Introduction to Game Design, Prototyping, and Development*, Addison-Wesley Professional.

Goodwin, S. (2016) *Polished Game Development*, Apress.

Guntupalli, R.C.C. (2008) *User interface design: methods and qualities of a good user interface design*.

Hipple, R. (2017) *Unite Austin 2017 - Game Architecture with Scriptable Objects - YouTube* *https://www.youtube.com/watch?v=raQ3iHhE_Kk&feature=youtu.be*, available: [accessed 11 Jan 2021].

Itch.io (2021) *Most used Engines* *https://itch.io/game-development/engines/most-projects*, itch.io, available: [accessed 11 Jan 2021].

Jacko, J.A. (2009) *Human-Computer Interaction. Interacting in Various Application Domains: 13th International Conference, HCI International 2009, San Diego, CA, USA, July 19-24, 2009, Proceedings, Part IV*, Springer Science & Business Media.

Koster, R. (2013) *Theory of Fun for Game Design*, Sebastopol: O'Reilly Media, Incorporated 1-4493-6321-0.

Lambottin, S. (2012) *The Fundamental Pillars of a Combat System*, available: https://www.gamasutra.com/view/feature/175950/the_fundamental_pillars_of_a_.php [accessed 30 March 2021].

Lim, C.-U., Baumgarten, R. and Colton, S. (2010) 'Evolving behaviour trees for the commercial game DEFCON', *European conference on the applications of evolutionary computation*, 100–110.

Rasmussen, J. (2016) *Are Behavior Trees a Thing of the Past?* *https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php*, available: [accessed 11 Jan 2021].

Sabbagh, M. (2015) *How to design formidable and unforgettable video game enemies* *https://www.gamasutra.com/blogs/MichelSabbagh/20151021/256899/How_to_design_formidable_and_unforgettable_video_game_enemies.php*, available: [accessed 11 Jan 2021].

Simpson, C. (2014) *Behavior trees for AI: How they work* *https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php*, available: [accessed 11 Jan 2021].

Sinicki, A. (2017) *Learn Unity for Android Game Development*, Apress.

Stonehouse, A. (2014) *User interface design in video games*, available: https://www.gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games.php [accessed 11 Jan 2021].

Sweetser, P. and Wiles, J. (2002) 'Current AI in games: A review', *Australian Journal of Intelligent Information Processing Systems*, 8(1), 24–42.

Toftedahl, M. (2019) *Which are the most commonly used Game Engines?* *https://www.gamasutra.com/blogs/MarcusToftedahl/20190930/350830/Which_are_the_most_commonly_used_Game_Engines.php*, available: [accessed 11 Jan 2021].

Unity, T. (2021) *Unity-Technologies/2d-extras*, available: https://github.com/Unity-Technologies/2d-extras [accessed 11 Jan 2021].

Unity Technologies, U. (2019) *Unity QA - LTS Releases* *https://unity3d.com/unity/qa/lts-releases*, available: [accessed 11 Jan 2021].

Unity Technologies, U. (2020) *Optimize performance of 2D games with fewer GameObjects, colliders, batch calls & more \textbar Unity* *https://unity.com/how-to/optimize-performance-2d-games-unity-tilemap*, available: [accessed 11 Jan 2021].

Vossen, B. (2015) *Enemy design and enemy AI for melee combat systems* *https://www.gamasutra.com/blogs/BartVossen/20150504/242543/Enemy_design_and_enemy_AI_for_melee_combat_systems.php*, available: [accessed 11 Jan 2021].

Wang, H. and Sun, C.-T. (2012) 'Game Reward Systems: Gaming Experiences and Social Meanings'.

Wang, X. (2016) 'Game Design Patterns for CPU Performance Gain in Games'.