Game mechanics development project report (CMP302)

Terminology

“Buffs” are temporary or permanent positive effects on game objects. Traditionally they can vary from healing the player to making movement easier or more effective.

“Debuffs” are temporary or permanent negative effects on game objects. Traditionally these can vary from a wide range of effects like making vision or movement difficult.

Summary

This mechanic is a potion system that changes player gameplay in significant ways. A wide range of games have this, from RPGs to FPSs. The concept in games can be traced back to the creation of Dungeons&Dragons roleplaying system in the seventies. They can be both drunk or throwed.

1. Multiple potions can be used, including:
    a. “Movement potion, drink it to increase player movement speed, temporarily; throw it to stun enemies in a circular area of impact.
    b. “Dash potion”, drink it to enable the player to dash, temporarily; throw it and the potion explodes doing damage on a radius.
    c. “Invincibility potion”, drink it to avoid the player from taking damage temporarily; throw it to create a circular area that blocks projectiles on impact.

3. Potion holding system:
    a. Player cannot hold more than one potion at a time and can only use the abilities of potion being held.

4. User Interface toolbar for player health and potion which is being held

A video demonstrating the project can be seen here:
https://youtu.be/EuGq3g3Ci9o

Requirement Specification

Introduction

Purpose:

The use of potions is widespread among all genres of games. This project aims to create a simple system that can be implemented and expanded in any game using the same engine. This is an investigation of how such a system can be created.

The components delivered of the project include a potion and health system. All aspects of the game will be viewed from a top down angle.

Overall Description

Product Perspective

The main objective of the project is creating a working potion system, where each potion will change the interaction with the player avatar in a significant way. There is a proper 2-dimensional area for testing this system.

Product Functions

1. Allow player to move and collide with potions to pick them up
2. Allow player to use the potion held and activate their properties
3. Allow player to see which potion is active through the toolbar UI
4. Allow other game developers to derive from the already created potions to implement new content (such as potions or abilities)
5. Allow a designer to change a set of data related to all potions effects without deep knowledge of programming

User Classes and Characteristics

This project is built having in mind the collaboration of software engineers and game designers who are familiar with most game engines.

Godot engine was used and it has the advantage that it can be easily used for non-programmers if set up properly. But engine knowledge allows designers to create new complex features.

Design and Implementation Constraints

The project was planned and done from scratch, hence not based on previous projects and no code from the internet was used.

Since there is no artist assigned to the project, the developer got assets from OpenGameArt for the game's environment and characters (all have the proper licenses for this use).

System Features

Potion: Movement

Description:
The potion can be taken from the test area. When you drink it, you get temporarily faster movement speed. If you throw the potion it will explode where you clicked with the mouse, enemies in the area will be temporarily stunned.

Priority: 8

Stimulus / response consequences:
For gameplay, the potion is accessible by moving on top of it and throwing or drinking is accessible by mouse.
For development, everything is done inside the Godot Engine. The potion has a scene for its object and a script for the logic.

Functional requirements:

REQ-1: Potion Collected
Potion must be collected by collision to be picked up.

REQ-2: Potion being held
If a potion is being held, a held potion is discarded and a new one is added.

REQ-3: Drink input
Player must fire an input to “drink” in order to activate the potion drinking properties.

REQ-4: Throw input
Player must fire an input to “throw” in order to activate the potion throw properties.

Potion: Dash

Description:

The potion can be taken from the test area. When the player drinks it, it gets the ability to dash temporarily. If you throw the potion it will explode where you clicked with the mouse, enemies in the area will be damaged.

Priority: 7

Stimulus / response consequences:

For gameplay, the potion is accessible by moving on top of it and throwing or drinking is accessible by mouse.

For development, everything is done inside the Godot Engine. The potion has a scene for its object and a script for the logic.

Functional requirements:

REQ-1: Potion Collected

Potion must be collected by collision to be picked up.

REQ-2: Potion being held

If a potion is being held, a held potion is discarded and a new one is added.

REQ-3: Drink input

Player must fire an input to “drink” in order to activate the potion drinking properties.

REQ-4: Throw input

Player must fire an input to “throw” in order to activate the potion throw properties.

Potion: Invincibility

Description:

The potion can be taken from the test area. When the player drinks it, it becomes invulnerable to damage. If the player throws the potion it will explode where clicked with the mouse, creating a circular area that blocks enemy shoots.

Priority: 7

Stimulus / response consequences:

For gameplay, the potion is accessible by moving on top of it and throwing or drinking is accessible by mouse.

For development, everything is done inside the Godot Engine. The potion has a scene for its object and a script for the logic.

Functional requirements:

REQ-1: Potion Collected

Potion must be collected by collision to be picked up.

REQ-2: Potion being held

If a potion is being held, a held potion is discarded and a new one is added..

REQ-3: Drink input

Player must fire an input to “drink” in order to activate the potion drinking properties.

REQ-4: Throw input

Player must fire an input to “throw” in order to activate the potion throw properties.

Test Map

Description:

All the potions and enemy will be on a 2 dimension top down viewed area.

Priority: 6

Stimulus / response time:

For gameplay, the player can explore the area but can't leave it.

For development, the area will be open on Godot Engine editor. The test area has a scene for its object, a script for the logic and all other elements of the game contained inside of it.

Functional Requirements:

REQ-1: Player

The test area must have a player to interact with the gameplay mechanics

REQ-2: Enemy

The test area must have an enemy to test all potions.

REQ-3: Potions

The test area must have potions for gameplay testing

REQ-4: Enclosed area

The area must be closed so the player cannot leave this area, since this is a test level.

Toolbar UI

Description:

The toolbar UI must show which potion is collected, player health, keybinds for drink and throw, the keybind for activating and deactivating the enemy, the bars of active potion effects with its duration.

Priority: 9

Stimulus / response sequences:

For gameplay, the player will use the keybinds shown on the UI to move, drink and throw the potions, activate temporary abilities (ex. Dash with “space bar”) and activate or deactivate the enemy (key “1”).

Depending on what potion is currently collected, the potion symbol will change and it shows the effect of drinking or throwing the potion.

Functional Requirements:

REQ-1: Display collected potion

Displays which potion has been picked up.

REQ-2: Display appropriate keybinds

Display the appropriate keybinds for drink, throw and activate the enemy, showing the player how to use them.

REQ-3: Show the correct amount of player health

The toolbar UI must display the correct amount of player health, changing the bar when the player takes damage.

REQ-4: Show current active potion buff effect and its duration

The toolbar UI must display the potion buff effect active on a bar that decreases to show how long the duration of the buff will last.

Player character

Description:

The player character must be able to move around the level according to the player input. The player must also be able to pick up and use potions.

Priority: 10

Stimulus / response sequences:

In game, the player character will respond to the correct movement keys (W/A/S/D) to move around the scene. Space bar to dash(when available) and the mouse to drink (right button) or throw (left button) a potion.

Functional requirements:

REQ-1: Respond to input

The player character must respond to input to move and use potions.

Enemy character

Description:

The test area will contain one enemy for testing the potions. The enemy can be activated or deactivated, stopping or starting its movement and shooting capacity. When active it moves randomly and shoots in the player direction.

Priority: 8

Stimulus / response sequences:

In game, the enemy can take damage from the player, which has a visual response but does not affect gameplay.

Pressing the key “1” will toggle the activation of the enemy logic.

Functional requirements:

REQ-1: Respond to damage

The enemy visual effect must show when taking damage.

REQ-2: Shoot the player

The enemy can shoot the player every few seconds.

REQ-3: Move

The enemy can move randomly inside the test area

REQ-4: Toggle logic

The enemy must be able to be activated and deactivated by pressing the key "1"

Other non-functional requirements

Performance requirements

While the game is running, a stable frame rate for the game elements to function correctly is essential. A low frame rate could cause effects like throwing being very imprecise, when it should be a smooth animation. A low frame rate will always cause a bad user experience, like delay to input by the player.

Software quality

High quality software implementation means lots of time saved for the creator of the code and other collaborators. Therefore this work keeps that in mind, object-orientation coding with proper coding standards ensures an easy to understand and easy to change software.

Every step of the project was subjected to testing and iteration to ensure everything works as intended.

Method

Potion selection

The main pillar of the game is an easy to use potion system. The player can simply collide with the potion on the test area to collect them. When the player gets a new potion a visual signal will show which is selected.

Potion data

All potions have the same base logic, making it easy to implement new elements and easily change attribute. Since several potions change player attributes we set a reference to the player in all potions, this happens on the method setup() of each potion, called when a potion is collected

When a player wants to drink a potion he/she emits a signal connected to the drink() method of the potion he/she might be holding. This modular design allows potion logic to be more versatile and easily configurable.

When a player wants to throw a potion the logic is similar to the one described above. The difference is that the potion enters the state of being throw after calling the method throw(). This logic is followed in the _process() method, which is the update method from Godot engine objects that is called every frame. If the potion is in state “being thrown”, it will be deleted on contact with a wall or explode when reaching the target location. If it reaches the target location (where the player clicked with mouse), the method explode() is called and does the potion effect depending on its type.

Potion activation

After the player collects a potion the mouse right button or left button click allows to drink it or throw it. If thrown the potion goes to the mouse position to do its effects or is destroyed when colliding with the wall.

Potion: Dash

The dash potion drink effect gives the player the ability to dash temporarily using the “spacebar” key. The dash script simply changes the player object boolean value that allows it to dash or not.
When dashing the position of the mouse is used to know which direction the player dashes to. The speed is changed to a temporarily big speed value that decreases fast, when it reaches zero the player returns to normal speed and thats how the dash effect is achieved .
The dash potion throw effect causes an explosion on the target location doing damage to the enemy if the enemy is on the circular radius of impact, the radius and damage are sent by the potion object even though the damage variable is not used.
When the potion is throw its state changes to “being thrown”, this sends a signal to the map scene where the explosion effect is created. The map scene creates a new instance of the scene “explosion” which does the effect depending on the potion type.

Potion: Invincibility

The invincibility potion drink effect gives the player a temporary immunity to damage. The invincibility script sends a signal to the player object that changes a boolean variable that allows the damage function to have effect or not.

The invincibility potion throw effect creates a temporary wall on the target position which blocks enemy shoots.
When the potion reaches its destination (or collides with the boundaries of the test map), it emits a signal for the map scene. The map scene creates a new instance of the object “barrier” on the target position. This object is a static body and works like a wall of the test map. When created the barrier object will start a timer and delete itself after the timer ends.

Potion: Movement

The movement potion drink effect gives the player a temporarily increased movement speed. The movement potion script sends a signal to the player object that changes the speed variable temporarily by activating a timer.

The movement potion throw effect causes an explosion on the target area and stuns the enemy temporarily.
When the potion reaches its destination (or collides with the boundaries of the test map), it emits a signal for the map scene. The map scene creates a new explosion scene instance on the given position. The explosion lasts just a few moments (a fixed attribute but that could easily be changed), and while active checks if an object of the group “enemy” enters its collision zone. If so, it calls the take_damage() method on the enemy, passing as argument the potion specified damage value.

Toolbar UI

The toolbar UI is an object of type canvas that goes over the elements of the test map.
The health bar element is an element of type ProgressBar with an object of type label on top of it to display text.
The logic of the progress bar is handled by the engine, hence it is only necessary to access the scene of the UI and pass values to the bar and label.
When the player’s health changes, it sends a signal with the current health value to the test map which in turn sends the updated health value to the UI.
The text for the potions type and effects is set and changed by the test map scene when the method create_potion() is called. This happens everytime a potion is picked up, used or the player dies.
The potions buffs bar are created when the player drinks the potion. The potion scene sends a signal to the UI scene with the duration, color and text of the bar.
The game controls are just always there shown on the UI, but the buttons use potions and are sensitive to context whenever the player is holding a potion.

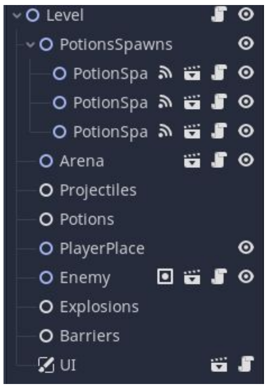
Enemy

The enemy logic is simple, it can be active by moving randomly (get a random position from the new_position() method) and shooting the player (using a timer method).
When the enemy object is deactivated the boolean variable “active” wont allow for the method _process() and method _on_Timer_timeout() to work. If the enemy is stunned, the method stun() will toggle the boolean variable stunned which works the same as the active variable, the only difference is that it has a timer to make the stun effect temporary.
The method setup() will organize the references the enemy object needs.
The method damage() calls the animation player to show the damage effect.
The player_died() method just makes the player reference null to avoid bugs.

Player

To move the player uses _process() method which handles the player movements by getting the input from movement keys and updates all other aspects of movement, when a player dashes it will use the appropriate speed vector.
Whenever the player takes damage the take_damage() method runs the damage animation and sends the signal to update the UI health.
To handle the throw and drink actions the throw(), drink() methods are used. They send the signals for potion mechanics to the test map which in turn sends them to appropriate functions.
The _input() method is a callback that is called when a user presses the configured keys and in this case we use it to call the methods throw(), drink() or dash().
Whenever the player's health reaches zero the method die() is called emitting a signal to avoid bugs in the rest of the code related to player death and deletes the player instance.

Test Map



The test map has the “tree” structure shown above. In godot the engine organizes everything in nodes, which are the elements shown on the image above.The position of each node on the scene matters for visual hierarchy, the top one is always drawn below the one below.

The potion spawn nodes are the positions where each potion is placed. This object does collision with the player to then create the potion.

The arena node is where the wall objects are placed.

The projectiles node stores projectiles when they are created. Projectiles only do collisions and call functions of other objects for the results of this collision.

The potions node stores the potion objects when they are created. Potions have many functions as described before.

The player place node is simply the initial location of the player and the position when the player respawns.

The enemy node is the enemy object and has many methods as described before.

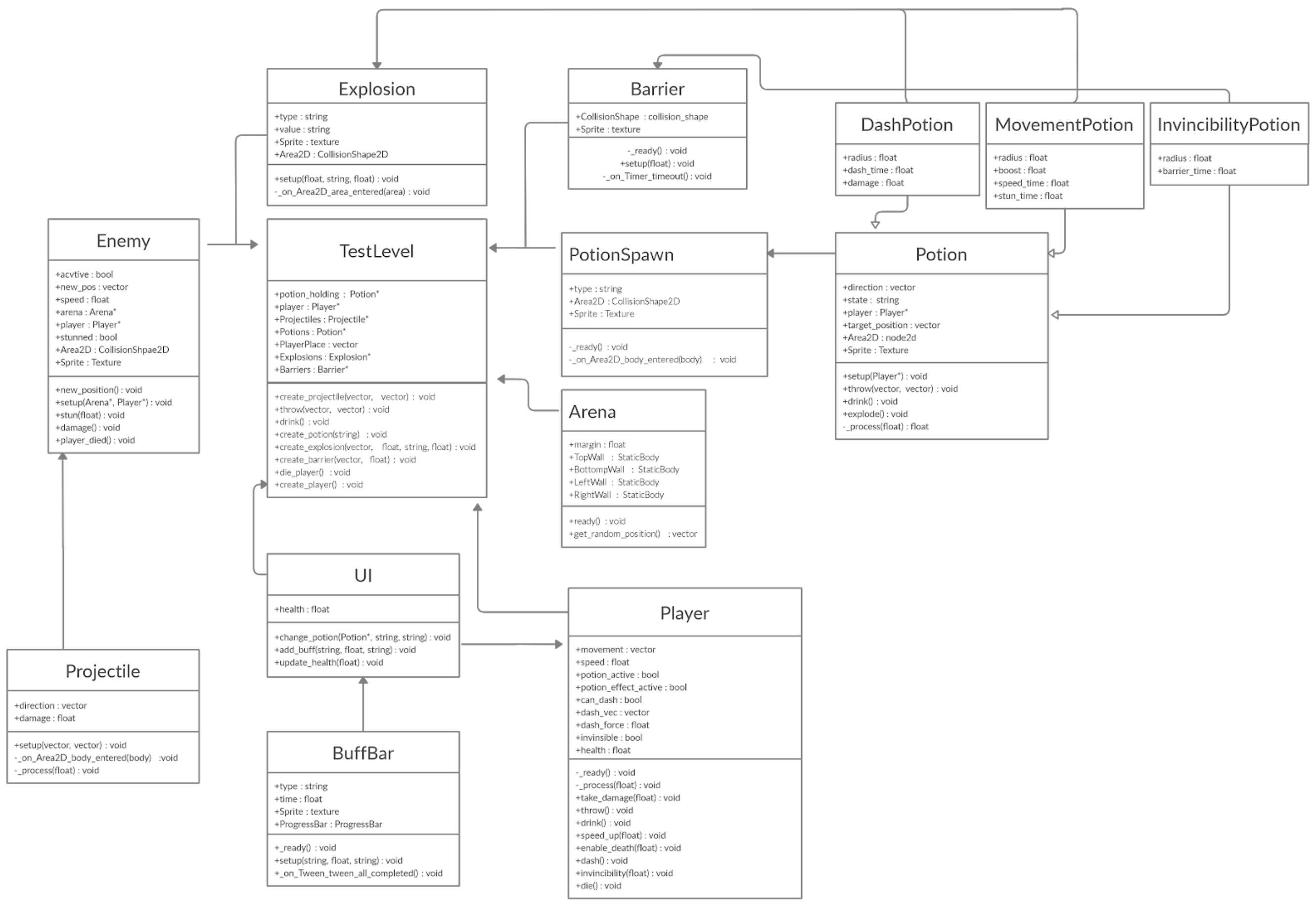
The explosions node stores all explosions that are created. These explosions in turn do their effects on the enemy or player.

The barrier node stores all barriers that are created by the invincibility potion.

The UI node is the object UI which has all UI elements.

Development

UML Diagram



Technical Discussion:

The godot engine architecture works with an abstraction called scenes. Scenes work as classes, objects or traditional scenes (several objects together). A scene is structured as a tree of nodes. Nodes are objects pre-build or instances of other scenes, furthermore they can have scripts connected to them, which are pieces of code attached that can run various types of methods, callbacks, variables, etc. The project is also structured as a tree, you have parent and child nodes that have access to each other by godot functions.
Using this architecture each game object will have a scene, like the player, and this facilitates to construct the final game scene since some are already instanced (enemy) and others can be instanced dynamically (projectiles).
However, for a node to access another node through the hierarchy is not ideal, especially when trying to access a node above it on the tree since the structured order above it can change. As a solution Godot has “signals”. These are information which nodes can send to other nodes, sometimes an intermediate node that has access to both helps, doing therefore a connection. This way the player node can send a signal when dying to the level node which in turn will connect the signal to any relevant node that needs to know if the player died.

Development :

First nodes created were the player, enemy and test map. Both the player and enemy needed movement and collision to prepare the potions tests and the test map needed walls with proper collision to stop undesirable movement.

Once the area was ready to handle most potion effects, potions were created. The potions demanded new changes to the player and enemy. The enemy needed to react to damage, the player to show the picked up potions and get affected by its properties. Potions also required a visual effect both when being thrown and when being held (UI symbol). Movement potion was the first because it seemed simpler, the dash potion was second done since it is the most similar in terms of mechanics to the movement potion and the invincibility potion was left last because it creates a barrier making it the most distinct.

Once most core mechanics were finished, the UI was refined to be clear with proper texts and images showing potion effects, potion duration and player health.

Most difficulties came from bugs related to other nodes needing more information. Like in the example cited above of the player death, many times other elements depended on getting information to proceed with their intended purpose. The Godot signals feature was very useful in solving all problems of communication between nodes, especially because the test map node had access to most nodes.

Conclusion:

Planning the structure of the entire game ahead showed to be valuable any time I had doubts about what path to follow on development, like in which order to do the mechanics or how to structure the nodes.

Godot engine showed itself to be very intuitive and practical, in this scale of project it is not possible to feel its shortcomings in features compared to unreal or unity. This advantage and the node system showed how having an intuitive structure, in some aspects, made the whole process faster compared to other engines, even the learning was faster then what I took for unreal and unity.

Godot made it easy to create other potions after the first one was done because I could just copy the same node structure and script for the new ones. Another big thing is how easy it is to export variables for easy changes in design by anyone. On the practical side, Godot has the scripting tool and documentation inside the engine, making everything faster, but probably not as powerful as having the visual studio IDE, like in unreal. Through the game documentation it was a good experience planning the game and testing whenever a new object was created. The debugging tool complemented every single step of the process by helping find each bug and unused variables.

References:

https://opengameart.org/ for art,

https://docs.godotengine.org/en/stable/getting_started/step_by_step/your_first_game.html for Godot Engines basics on game developing,

https://docs.unrealengine.com/en-US/Engine/UI/index.html and

https://docs.unrealengine.com/en-US/Engine/QuickStart/index.html

For planning the basics of the game