Festive Fusion: A 2D Game

Trinath Pittala
Clemson University
School of Computing
tpittal@g.clemson.edu

Utkal Sirikonda
Clemson University
School of Computing
usiriko@g.clemson.edu

Geethakrishna Puligundla
Clemson University
School of Computing
gpuligu@g.clemson.edu

ABSTRACT
In this paper, we explain the design patterns and implementation details used to build our project Festive Fusion Game. The game is built robustly with user friendly and seamless functionality of saving, loading, restart and leaderboard with the help of creational patterns like Factory, Singleton, and a behavioral pattern - Memento. We go through important code snippets to explain how we implemented some critical parts of the game.

Categories and Subject Descriptors
D.2.11 [Software Architectures]: Patterns – Design patterns for game development. K.8.0 [Personal Computing]: General - Games.

General Terms
Algorithms, Design, Human Factors.

Keywords
Design Patterns, Software Design, Pygame, Python, 2D game, Top Down, RPG.

1. INTRODUCTION
The Festive Fusion [1] game is a holiday theme based game developed using Pygame [9] that displays the use of Software Design Patterns to achieve loosely coupled, clean code to build complex features. The use of generally used patterns makes source code easy to read and understand the way game elements get created and interact with each other. We mainly discuss 3 Patterns involved (2 Creational and 1 Behavioral) [8]. We also talk about how we used the basic vector method for enemies chasing the player functionality.

The paper is structured to provide a comprehensive understanding of the design, implementation, and lessons learned, serving as a resource for developers interested in bringing software design patterns to game development.

2. RELATED WORK
This game is inspired from Pokémon Leaf Green [7], a classic 2D top-down adventure game known for engaging, explorative game play. The game design is adopted from an adventure game tutorial by Alexander Farrell [4] and improved with Pokémon style interface.

To create maps for different levels we leveraged Tiled map editor [2]. This tool is used to design custom maps by embedding different sprites. The sprites Involved in creating game environments are taken from OpenGameArt [2].

Pygame [9] is a Python library for basic game development offering an essential API methods like colliderect (detects collision), clipline (clips line between given points), etc., easing the developer's effort.

Integrating and utilizing these tools and resources enabled us to build the game functionality in depth.

3. GAME OVERVIEW
In the game, the protagonist is exposed to different themed environments (levels) and enemies of different forms. The goal of the player is to find and reach home securely in each level, while defeating enemies and possibly catching collectibles to score points. To recover from damage taken from enemies, the player can find and collect Health potions, which are different for each level. The environments are based on Thanksgiving, Halloween and Christmas respectively, in increasing complexity to reach the goal.

3.1 Thanksgiving - Level 1
Level 1 is based on Thanksgiving, which acts as a basic level to make the player accustomed to the game. The environment will be open map and contains Turkey feast acting as health potions to recover health, and the enemies are evil Turkeys. The player can kill Turkeys and collect chest to gain score points. The player can also avoid action and just focus on reaching home, which is the goal.

3.2 Halloween – Level 2
Level 2 is based on Halloween. It is also an open map style and the level is little bit harder compared to first level with increased enemies. There are ghosts in different areas which act as enemies, and there are blood potions to recover health. The player can collect Pumpkins which is the collectible item in level 2.

3.3 Christmas – Level 3
The Christmas level is built as a maze, to have an added challenge of finding home. The environment is restricted in space, constraining the player from running far from Snowman, who are the enemies. The player can catch Ginger Breads along the way to recover health and collect Christmas Gifts as collectibles to increase score.

4. DESIGN AND IMPLEMENTATION
The game theme is inspired from Pokémon Leaf Green [7], which forms the basic layout. The player character is Charmander, a fire-throwing Pokémon. The player's attack is performed by left mouse click on enemies, and a fire ball is thrown in the direction of Charmander towards enemy.

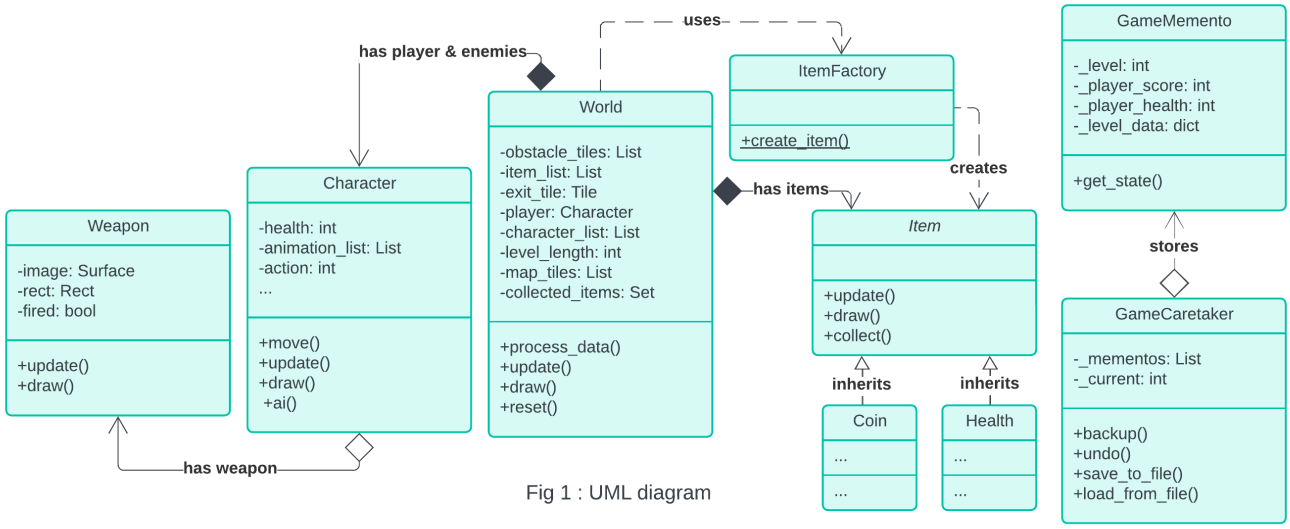

Fig 1 : UML diagram

In Fig1, we showed the UML Class diagram [10] of the game representing various classes and its relationships.

4.1 Game Mechanics

4.1.1 Attack

When the player comes into enemy attack range, the enemy starts chasing the player and tries to attack. The enemy attacks the player as they come in contact and the player's health decreases by 10% for each attack. The enemy's movement speed is set to half of player's speed to let the player have an option to escape.

4.2.2 Scoring

Catching a collectible item awards the player 10 points and defeating an enemy awards 100 points. The awarding is intentionally set to award more when player defeats enemy. Since the player can easily evade enemies and collect items to score points, the collectible items have lower weightage.

4.2 Classes in Game

4.2.1 World – Singleton Pattern

The World class defines elements such as level map, characters, and items (collectibles and health recovering items) for each level.

```
class World():
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance.initialize()
        return cls._instance

    def initialize(self):
        # Initialize all the attributes that were previously in __init__
        self.obstacle_tiles = []
        self.item_list = []
        self.exit_tile = None
        self.player = None
        self.character_list = []
        self.level_length = 0
        self.map_tiles = []
        self.collected_items = set()
```

Fig 2: World class implements Singleton pattern

World class reads csv file containing tile image names and initializes instances of elements in the level.

When we start the game, the world object is initialized once and for each level, we use the update member functions to change the values.

4.2.2 Character

The character class is responsible for building the player's character (Charmander) and enemies of each level. The class implements draw (display character on screen), move (translates character), update (updates health, alive status of character), functionality. The enemy characters have an ai method implemented, which defines their movement.

4.2.3 Item – Factory Pattern

Item class creates two kinds of items in each level: Collectible items (coins) and Health Potion items. Here we are using Factory method to create multiple items using the same interface. As seen in Fig 1, the Item Factory class takes item type as its parameter and decides the kind of item to create. The Item class is an abstract class with two abstract functions: update, collect and a draw method to draw the item. There are two concrete classes: Coin, Health Potion which inherit Item class and implement the above abstract methods to increase health when Health Potion item is collected and increase score when Coin item is collected. The code snippet for this pattern is not shown here for the sake of brevity.

4.2.4 State – Memento Pattern

The game offers save, and load feature to let the player stop and play after a while. The functionality is implemented with Memento pattern which maintains important information like Player's score, health, position, collected items, killed enemies and the level he is in. From the Fig 1, we can see that the Game Memento class holds the state variables need to save the state of game, and Game Caretaker class contains list of memento instances. Every time the player saves, current state of game is saved into a new instance of Game Memento and is appended into mementos list inside Game Caretaker class. Also, the current state is saved into a Json file. The load game reads from Json file,

creates a memento instance from its state values and returns the instance.

4.3 Enemy Chasing Algorithm

The enemy characters chase the player when the player comes into attack range of enemy character. The enemy's movement is dependent on both enemy's and player's position in every instance. The code snippet provided in Fig 3, describes the enemy character's movement pattern.

The enemy's line of sight is a line between player and enemy and we take help of Pygame's clipline method to check if any of the

```
# create a line of sight from the enemy to the player
line_of_sight = ((self.rect.centerx, self.rect.centery), (player.rect.centerx, player.rect.centery))
# check if line of sight passes through an obstacle tile
for obstacle in obstacle_tiles:
    if obstacle[1].clipline(line_of_sight):
        clipped_line = obstacle[1].clipline(line_of_sight)


#check distance to player
dist = math.sqrt(((self.rect.centerx - player.rect.centerx) ** 2)  + ((self.rect.centery - player.rect.cent
if not clipped_line and dist > constants.RANGE:
    if self.rect.centerx > player.rect.centerx:
        ai_dx = -constants.ENEMY_SPEED
    if self.rect.centerx < player.rect.centerx:
        ai_dx = +constants.ENEMY_SPEED
    if self.rect.centery > player.rect.centery:
        ai_dy = -constants.ENEMY_SPEED
    if self.rect.centery < player.rect.centery:
        ai_dy = +constants.ENEMY_SPEED
```

Fig 3: Enemy chasing code snippet

World's obstacle tiles lie between the player and enemy. If no obstacles are on the way and player is in attack range of enemy, the enemy is displaced towards the player. Hence, the enemy tries to reach and attack the player as they are in contact.

5. CONCLUSION

This paper presented the development of 2D game with Pygame using design patterns: Singleton, Factory, and Memento to achieve modular code. The game address 3 themes covering fall & winter holiday festivals, with increased levels of difficulty. A custom enhanced enemy chasing algorithm was designed to provide dynamic, challenging player-enemy interactions.

Through this project, we showcased how lightweight frameworks like Pygame can be leveraged to create games while incorporating advanced software design principles.

In future, the game can be extended to leverage Observer pattern since it has a lot of event handling. The hardcoded values like menu positions, resolution and scaling can be made dynamic according to the screen resolution of user. This work contributes to understanding of combining game design with software engineering principles.

6. REFERENCES

[1] https://github.com/Lasterminator/Festive-Fusion, GitHub Repository of the project.

[2] https://www.mapeditor.org, A Map Generation software.

[3] https://opengameart.org, Open source Game assets store.

[4] https://www.youtube.com/watch?v=w8i5sizgVNs&list=PLn8cgfOA2qzeXxXe7DBWxX1UR5fJ4MDTb, a tutorial to build a 2d top down game with Pygame.

[5] https://www.pixabay.com, A Multimedia open source store.

[6] https://www.uppbeat.io, A Copyright free music store.

[7] https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9mon_FireRed_and_LeafGreen_Versions, A Pokémon game wiki.

[8] https://refactoring.guru/, A Resource for Design patterns.

[9] https://www.pygame.org, Game development Library in Python.

[10] https://www.uml-diagrams.org/, UML diagrams study material.