Project 7

Angel Romero
OOAD-Professor Montgomery
4/20/2023

Project Name and Member(s)

Name:

BoxFight

Members:

- Angel Romero(Self)

Final State of System

The Final State of the System has been altered and reduced due to only working on it by myself and having no one else to work with. Some things that were kept were:

- Primary Functionality:
    - Player and enemy object
        - Observer Design
        - Decorator (Items that change the object's state
    - Scene implementation:
        - Main
        - Level
        - Starting Menu
        - Settings
        - Game Over
    - Other Objects that are tied to being observers for the primary objects and interface to integrate patterns

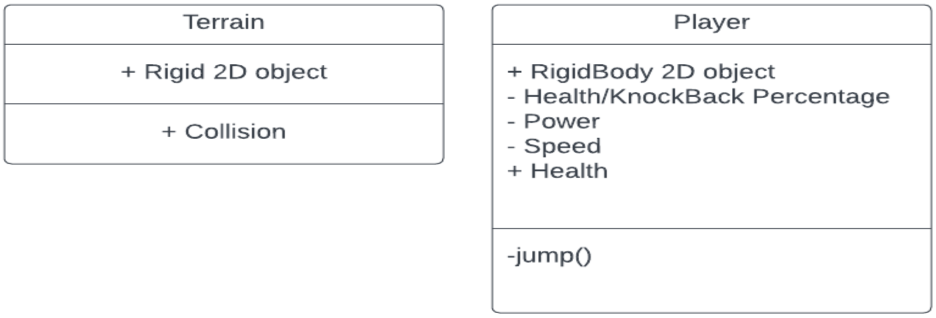
Other things that were not kept:

- Database:
    - Due to not having anything to record it on and the amount of work that goes on to develop the foundation of the game(no Unity experience) I have must drop it

- Differentiation in Level Design and Quantity:
  - I only developed one level and just because replicating a scene base has a lot of work put in just to make it different from the previous
- GUI:
  - Just making my game to reflect my skills from my OOAD class

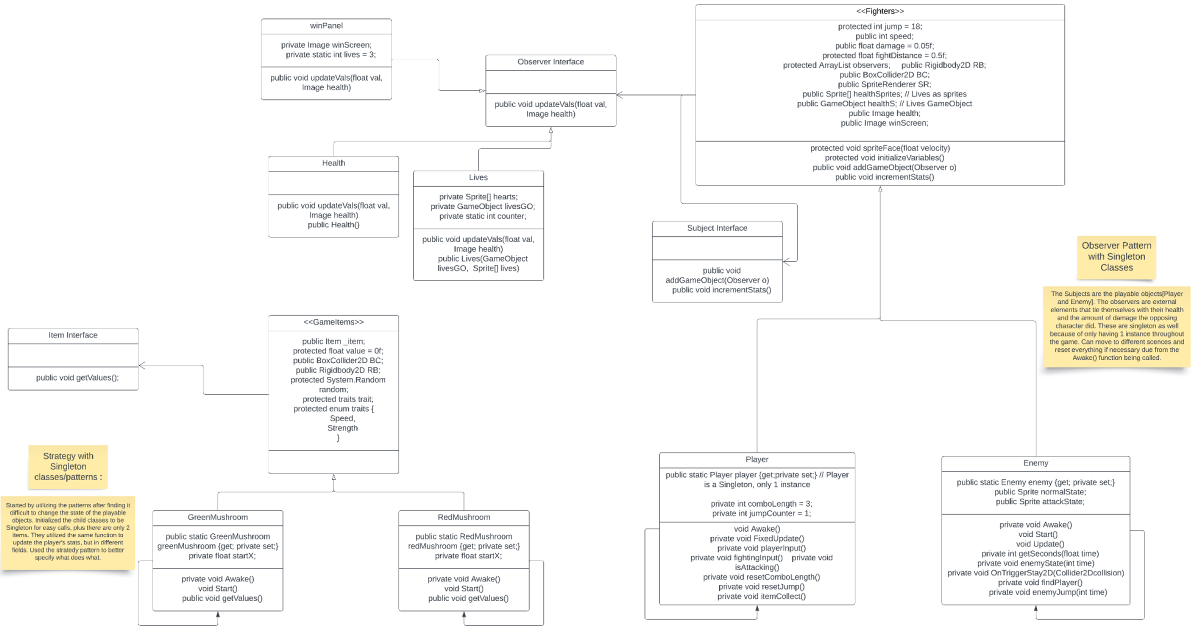
Final Class Diagram

Project 5 UML



Project 7 UML

https://lucid.app/publicSegments/view/cf6839c6-ef1f-42cf-9318-c2e10c33dbdb


winPanel
private Image winScreen;
private static int lives = 3;
public void updateVals(float val, Image health)
Observer Interface
public void updateVals(float val, Image health)
Health
public void updateVals(float val, Image health)
public Health()
Lives
private Sprite[] hearts;
private GameObject livesGO;
private static int counter;
public void updateVals(float val, Image health)
public Lives(GameObject livesGO, Sprite[] lives)
<<Fighters>>
protected int jump = 18;
public int speed;
public float damage = 0.05f;
protected float fightDistance = 0.5f;
protected ArrayList observers; public Rigidbody2D RB;
public BoxCollider2D BC;
public SpriteRenderer SR;
public Sprite[] healthSprites; // Lives as sprites
public GameObject healthS; // Lives GameObject
public Image health;
public Image winScreen;
protected void spriteFace(float velocity)
protected void initializeVariables()
public void addGameObject(Observer o)
public void incrementStats()
Subject Interface
public void addGameObject(Observer o)
public void incrementStats()
Observer Pattern with Singleton Classes
Item Interface
public void getValues();
<<GameItems>>
public Item _item;
protected float value = 0f;
public BoxCollider2D BC;
public Rigidbody2D RB;
protected System.Random random;
protected traits trait;
protected enum traits {
Speed,
Strength
}
Strategy with Singleton classes/patterns :
GreenMushroom
public static GreenMushroom greenMushroom {get; private set;}
private float startX;
private void Awake()
void Start()
public void getValues()
RedMushroom
public static RedMushroom redMushroom {get; private set;}
private float startX;
private void Awake()
void Start()
public void getValues()
Player
public static Player player {get;private set;} // Player is a Singleton, only 1 instance
private int comboLength = 3;
private int jumpCounter = 1;
void Awake()
private void FixedUpdate()
private void playerInput()
private void fightingInput() private void isAttacking()
private void resetComboLength()
private void resetJump()
private void itemCollect()
Enemy
public static Enemy enemy {get; private set;}
public Sprite normalState;
public Sprite attackState;
private void Awake()
void Start()
void Update()
private int getSeconds(float time)
private void enemyState(int time)
private void OnTriggerStay2D(Collider2Dcollision)
private void findPlayer()
private void enemyJump(int time)

The Subjects are the playable objects[Player and Enemy]. The observers are external elements that tie themselves with their health and the amount of damage the opposing character did. These are singleton as well because of only having 1 instance throughout the game. Can move to different scences and reset everything if necessary due from the Awake() function being called.

Started by utilizing the patterns after finding it difficult to change the state of the playable objects. Initialized the child classes to be Singleton for easy calls, plus there are only 2 items. They utilized the same function to update the player's stats, but in different fields. Used the stratedy pattern to better specify what does what.

Third-Party Code vs Original Code

I only used what Unity provided me and the packages that came it came with. This would compose of:

- Default Inheritance:
    - MonoBehaviour Class
- Default Functions(Not used all the time):
    - Void Start()
    - Void Update()
- Object Instantiation when creating objects
    - Example:
        - Creating an empty object that will need a Rigid 2D Object, BoxCollider, and whatever I choose to manipulate using my script class

Statement on the OOAD process for my overall Semester Project

The key design process elements or issues(positive or negative) that I have experienced in my analysis and design of the OO semester project:

- Spaghetti Code(Negative):
    - When I went into this project, I was overwhelmed with the amount of work I had to do:
        - Relearning C#
        - Learning how Unity works
            - Majority of my time was spent here!
        - Animation/GUI
            - Far less after I was told I didn't need to implement a GUI if I didn't have time. It was a lifesaver!
        - Functionality Goals
        - Structuring and ordering my assets
    - I ended up starting with spaghetti code which messed up my mojo due to me revisiting my code to restructure because of lack of

understanding and early implementations meant the project would be much easier to maintain and extend.

- Observer pattern (Positive):
    - I was trying to make a non-extensible game so that I can modify later, this wouldn't work. Many of the games elements relied on the implementation of the Observer pattern due to these elements change when the objects Interacted with one another. Made it so much more simpler to add a new object that is tied in with the primary game objects and observes it's state.
- Inheritance(Positive):
    - Again, I was trying to implement a broad and messy game to re-initialize and edit later but it was too much of a task to do. This was much more prevalent when it came to revisiting my code and not knowing what elements did the player and the enemy class object had in common. Abstract interface came in handy!
- Singleton Pattern (Positive)
    - Even though I wasn't able to create an overly structured and detailed code, I though having a singleton for a 1-player game was nice, as well as the enemy.
- Strategy Pattern (Neutral)
    - Trying to plug in a strategy pattern for the items proved difficult and it proved to be much more than the task at hand.

<u>Record your presentation! 10-15 minutes.</u>

The Presentation is in the Repo! Please Let me know if you cannot access it!