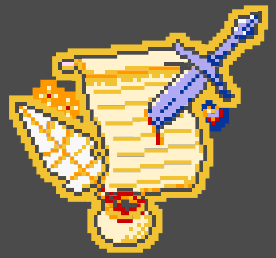


JAVEX (Game Engine)

DONE BY: AINSLEY CHANG, JOVAN LIM, VARSHA , LIN ZHENMING, YEOW DAO XING

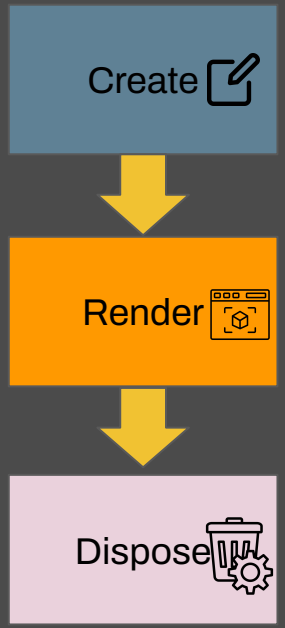
TABLE OF CONTENTS

01
GAME ENGINE

02
OOP CONCEPTS

03
Game Demonstration

SIMULATION LIFECYCLE



- Simulation Life Cycle
  - 3 main functions
    - Create
    - Render
    - Dispose

Create

- SceneManager()
- InputManager()
- OutputManager()

Render

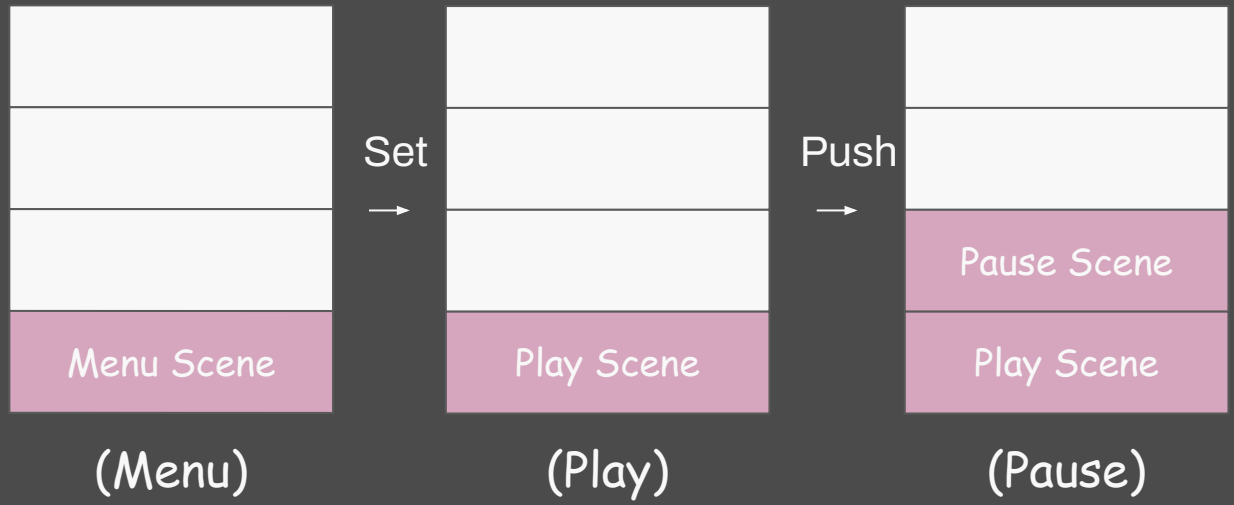
- Running of different Manager Classes
- I.e. Running of PlayScene in SceneManger

Dispose

- Disposal of memory once exit
- Memory Allocation for future use

SCENE MANAGER

- Stack system
  - Push()
  - Pop()
  - Set()
  - Peek()



INPUT/OUTPUT

- Boolean Variables
  - UpPressed()
  - DownPressed()
  - Etc
- Input Flexibility
- Output can be called easily in different scenes
- Changes are easily made by changing the file name

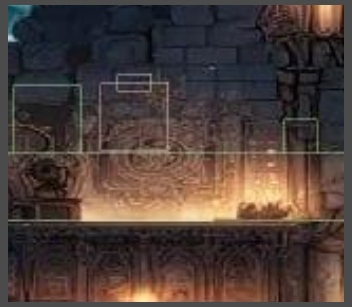
PLAYER CONTROL

UpPressed()
= True

applyfocetocenter()

- Reference from Input values
- Utilizes if conditions to apply logic
  - If(UpPressed) = True
    - Player body get jump statement to jump
  - Different input can be easily changed as boolean statement would remain the same and actions based on boolean values
  - Flexible to change

ENTITY MANAGER



Entity Manager Class Responsibility

- Creation
- Rendering

(To be called by)

PlayScene

- Uploading of creation to display on playscene by rendering Box2DR

AI CONTROL

LibGDX pre-built library

- Pre-Built Library
  - Arrive
  - Seek
  - Pursue
  - Etc

Seek

- Setting of 2 targets to be steerable
  - Implements Sterrable<Vector2>
  - Target (Player to be targeted to be followed)
  - Entity (Boss to follow 'Target')
- Implement calculations for Seek method to run calculations to follow user

'Target' to be "seeked"

'Entity' to have AiControlManagement implemented on it to follow

COLLISION MANAGER

```
public static final short PLAYER_BIT = 1;
public static final short ENEMY_BIT = 2;
public static final short ENEMY_HEAD_BIT = 4;
public static final short TERRAIN_BIT = 8;
public static final short REWARD_BIT = 16;
```

1 = 00001
2 = 00010
4 = 00100
8 = 01000
16 = 10000

(1) Player kill enemy = player_bit **OR** enemy_head_bit = 00101 = 5

```
Player land on enemy top
case Constants.PLAYER_BIT | Constants.ENEMY_HEAD_BIT:
    if (fixA.getFilterData().categoryBits == Constants.ENEMY_HEAD_BIT) {
        ((Enemy) fixA.getUserData()).hitOnHead();
```

***Note :**
- collision logics are defined and executed in CollisionManager.
- PlayScene listens to the collisions via the CollisionManager.

OOP Principles

1. Encapsulation - Correct use of access modifiers in all classes.
2. Inheritance - Extend common methods and attributes from a parent class.
3. Polymorphism - Usage of method overloading and overriding, for methods to have the same name but different implementations.
4. Abstraction - Separating the complex implementation of a code segment from its simple repeated usage elsewhere.

REUSABILITY 

1. Scenes & Scene Manager
   - ➢ To add a new scene, extend the Scene.java parent class and define the outlooks of it. Then, allow the SceneManager to .push(), .pop() or .set() the specific scene at any point of the player's journey.

2. Output Manager
   - ➢ To play a different music, simply call the outputManager.play(musicPath), where musicPath can be defined in each specific scene or specific collision.

SCALABILITY 

1. Input Manager
   - ➢ To accept more input device or input signals, configure it within the InputManager.java class and allow downstream classes to query the information from it.
2. Ai Control Manager
   - ➢ To equip more AI behaviours, define the AI behaviour(s) within the AiControlManager.java class, and pass the intended AI-controlled entities as its arguments to the constructor.
3. Collision Manager
   - ➢ To insert more kinds of collisions to be detected and resolved, simply add on to the .beginContact() method of the CollisionManager.java
4. Entity Manager
   - ➢ To spawn a new entity of any kind, configure the .initialise() method of the EntityManager.java class.

LIMITATION

1. Absence of different levels
    - Enemies, maps and design of everything are fixed with no variants to the game. Therefore, the scenes are unable to generate new variants as everything is preset

    - How to Overcome (in the future)
        - Create a new class called “Level.java”
        - Allow this parent class to spawn several child classes of different levels, each with their own configurations, e.g. (1) Win condition, (2) Number of enemies, (3) Time Limit
        - Each of this levels can have their own audio file to play as well as background image to render as well.

Game Demo