CSCI 205 Final Project: Design Manual

Introduction

Our project is an implementation of the popular mobile game Doodle Jump. Our goals when starting this project were as follows:

1. Recreate the game's visual style
2. Replicate the game's physics as closely as possible
3. Increase the difficulty as player progresses

In order to accomplish these, we used a heavily object-oriented approach and committed ourselves to documenting our changes as much as possible. Git branching, detailed commit titles, and the use of Gitlab's visual branch merge tool allowed us to stay organized and efficient in our development.

Our project structure is as follows:

```
Unset
ACTW
├── ChunkLoader.java  # Generates platforms in "chunks"
├── DifficultyState.java  # Enum to describe game
difficulty
├── GameState.java  # Enum to describe the state of the
game
├── Global.java  # Stores global variables'
├── MainApp.java  # Entry point for JavaFX
├── MovingMonster.java  # Class representing monsters
capable of moving
├── Platform.java  # Class representing individual
platforms
├── SceneManager.java  # Manages switching between JavaFX
scenes
```

```
├── StationaryMonster.java  # Class representing monsters
that don't move
├── charactersandmobs
│   ├── BlackHole.java  # Black hole mob
│   ├── DoodleChar.java  # Doodle character
│   └── Obstacle.java  # Obstacle mob
└── screens
    ├── GameScreen.java  # The game screen (scene)
    ├── StartScreen.java  # The start screen (scene)
    └── StatScreen.java  # The stat screen (scene)
```

Our project's entry point is `MainApp.java`, in which we initialize a `SceneManager` object to manage scenes we subsequently initialize via the `*Screen` classes using `getScene()` methods. Once in `SceneManager`, they are assigned a scene label, which is a string used to access them from `SceneManager`. It also allows us to switch between scenes in cases where the scene variables themselves are out-of-scope. For example, in `GameScreen`, when the player dies, `SceneManager.activateScene()` is called to activate/switch to the stat screen, despite the `StatScreen` object itself being initialized in `MainApp`.

`SceneManager` is the key to our screen functionality. It works by facilitating the addition and removal of scenes from its catalog. Scenes in this catalog can be "activated", meaning that `SceneManager` adds them to the JavaFX `Stage`. As a result, switching between scenes, adding new ones, and removing old ones can be done across classes and without "hard code".

Our `*Screen` classes are the heart of our codebase. Specifically, our `GameScreen` class. `GameScreen` uses other classes such as `ChunkLoader`, `Platform`, and `DoodleChar` to create levels and allow character movement. It also keeps track of the score and facilitates the transition to the `StatScreen` upon player death. This allows us to maintain an object-oriented design while keeping our core game logic in a central class.

CRC Cards: Below are our relevant CRC cards, detailing the responsibilities of our classes as well as contributors of development

Class	Responsibility	Contributor
BlackHole	Represents a type of obstacle in the game which is deadly to the doodle character; extends Obstacle	Alexa , Charlie
DoodleChar	Represents the doodle character controlled by the user; handles doodle movement	Charlie, William, Trevor
MovingMonster	Represents a type of obstacle in the game (monster) which can move horizontally across the screen; extends Obstacle	Alexa
Obstacle	Parent class to BlackHole, MovingMonster, and StationaryMonster to represent a generic obstacle	Alexa
StationaryMonster	Represents a type of obstacle in the game (monster) which cannot move; extends Obstacle	Alexa
GameScreen	Represents the main screen for gameplay; handles character movement and gravity and brings in platform and obstacle objects	Alexa, Charlie, William, Trevor
StartScreen	Builds and manages the start screen scene (title, play button)	William
StatScreen	Builds and manages the stat screen scene (title, score, play, menu)	Alexa, William, Charlie
ChunkLoader	Responsible for generating and managing a chunk of the game world. Chunks are sections of the game's background that load as the character moves upward	Alexa, Trevor
DifficultyState	Enumerator to represent the difficulty levels in the game	Trevor, Alexa
Direction	Enumerator to represent the direction of movement for the doodle character	Charlie
GameData	Class to store the current score of the game to be passed to the stat screen	Charlie, Alexa, William
GameState	Enumerator demonstrating what state the game is in	Alexa
Global	Represents global variables used acrpdd the project	William
MainApp	Main entry point of the program where the game is built and ran; starts the JavaFX app	William, Charlie
Platform	Represents a platform in the game that the doodle character can land on; either normal or bounce	Alexa, Trevor
SceneManager	Class for managing scenes added with corresponding String label; scenes can be activated, accessed, and removed	Willam

User Stories

We created three primary user stories to guide our implementation of features that the user interacts with during gameplay:

1) I want to be able to control the horizontal movement of the doodle character as it jumps from platform to platform.

2) I want to be able to navigate between screens using play and menu buttons.

3) I want to be able to track my score as the game progresses, and see my final score at the end of each game.

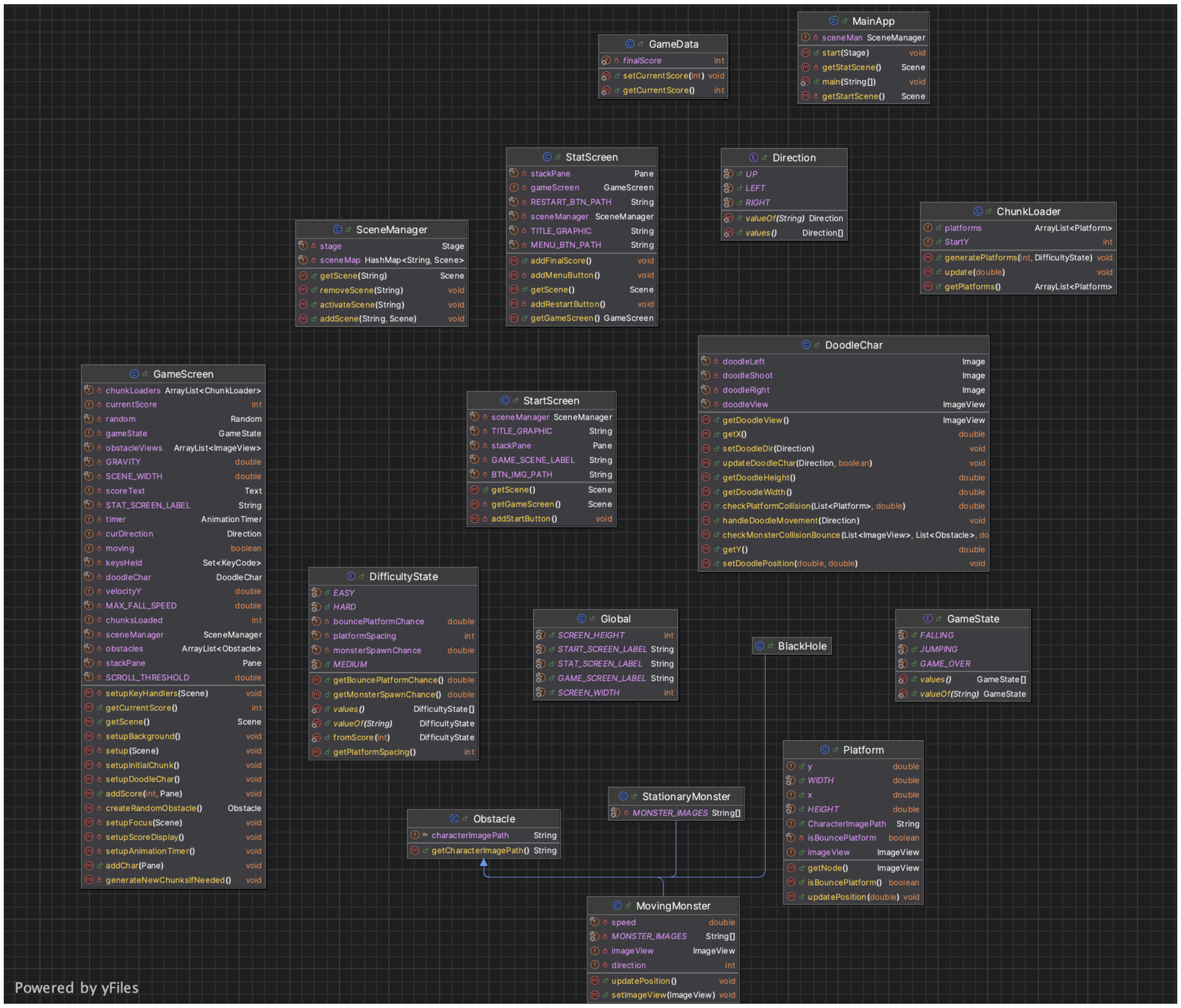
Object-Oriented Design

Our project utilizes object-oriented design via our usage of classes to manage different "objects" within our game. Such objects include obstacles, platforms, characters, monsters, and the game screens themselves.

Platforms and obstacles are loaded in "chunks" via the `ChunkLoader` class. Platforms are individual objects of type `Platform`, a class designed to stores positional, image, and other relevant data. Obstacles, such as mobs and black holes, have their own objects and utilize polymorphism. Our parent class, `Obstacle`, is an abstract class that `BlackHole` and `MovingMonster` extend. These chunk loader objects are utilized in `GameScreen` to create chunks as the player progresses through the level.

Our Scenes are managed via a `SceneManager` class. This class takes scenes generated from the various `*Screen` classes and stores them in a HashMap, mapping string labels to their respective scene counterparts. This allows the managing of scenes even when the scene variables themselves are out-of-scope.

Intellij Generated UML Diagram:


GameData
finalScore int
setCurrentScore(int) void
getCurrentScore() int
MainApp
sceneMan SceneManager
start(Stage) void
getStatScene() Scene
main(String[]) void
getStartScene() Scene
StatScreen
stackPane Pane
gameScreen GameScreen
RESTART_BTN_PATH String
sceneManager SceneManager
TITLE_GRAPHIC String
MENU_BTN_PATH String
addFinalScore() void
addMenuButton() void
getScene() Scene
addRestartButton() void
getGameScreen() GameScreen
Direction
UP
LEFT
RIGHT
valueOf(String) Direction
values() Direction[]
ChunkLoader
platforms ArrayList<Platform>
StartY int
generatePlatforms(int, DifficultyState) void
update(double) void
getPlatforms() ArrayList<Platform>
SceneManager
stage Stage
sceneMap HashMap<String, Scene>
getScene(String) Scene
removeScene(String) void
activateScene(String) void
addScene(String, Scene) void
DoodleChar
doodleLeft Image
doodleShoot Image
doodleRight Image
doodleView ImageView
getDoodleView() ImageView
getX() double
setDoodleDir(Direction) void
updateDoodleChar(Direction, boolean) void
getDoodleHeight() double
getDoodleWidth() double
checkPlatformCollision(List<Platform>, double) double
handleDoodleMovement(Direction) void
checkMonsterCollisionBounce(List<ImageView>, List<Obstacle>, do
getY() double
setDoodlePosition(double, double) void
GameScreen
chunkLoaders ArrayList<ChunkLoader>
currentScore int
random Random
gameState GameState
obstacleViews ArrayList<ImageView>
GRAVITY double
SCENE_WIDTH double
scoreText Text
STAT_SCREEN_LABEL String
timer AnimationTimer
curDirection Direction
moving boolean
keysHeld Set<KeyCode>
doodleChar DoodleChar
velocityY double
MAX_FALL_SPEED double
chunksLoaded int
sceneManager SceneManager
obstacles ArrayList<Obstacle>
stackPane Pane
SCROLL_THRESHOLD double
setupKeyHandlers(Scene) void
getCurrentScore() int
getScene() Scene
setupBackground() void
setup(Scene) void
setupInitialChunk() void
setupDoodleChar() void
addScore(int, Pane) void
createRandomObstacle() Obstacle
setupFocus(Scene) void
setupScoreDisplay() void
setupAnimationTimer() void
addChar(Pane) void
generateNewChunksIfNeeded() void
StartScreen
sceneManager SceneManager
TITLE_GRAPHIC String
stackPane Pane
GAME_SCENE_LABEL String
BTN_IMG_PATH String
getScene() Scene
getGameScreen() Scene
addStartButton() void
DifficultyState
EASY
HARD
bouncePlatformChance double
platformSpacing int
monsterSpawnChance double
MEDIUM
getBouncePlatformChance() double
getMonsterSpawnChance() double
values() DifficultyState[]
valueOf(String) DifficultyState
fromScore(int) DifficultyState
getPlatformSpacing() int
Global
SCREEN_HEIGHT int
START_SCREEN_LABEL String
STAT_SCREEN_LABEL String
GAME_SCREEN_LABEL String
SCREEN_WIDTH int
BlackHole
GameState
FALLING
JUMPING
GAME_OVER
values() GameState[]
valueOf(String) GameState
Platform
y double
WIDTH double
x double
HEIGHT double
CharacterImagePath String
isBouncePlatform boolean
imageView ImageView
getNode() ImageView
isBouncePlatform() boolean
updatePosition(double) void
StationaryMonster
MONSTER_IMAGES String[]
Obstacle
characterImagePath String
getCharacterImagePath() String
MovingMonster
speed double
MONSTER_IMAGES String[]
imageView ImageView
direction int
updatePosition() void
setImageView(ImageView) void
Powered by yFiles