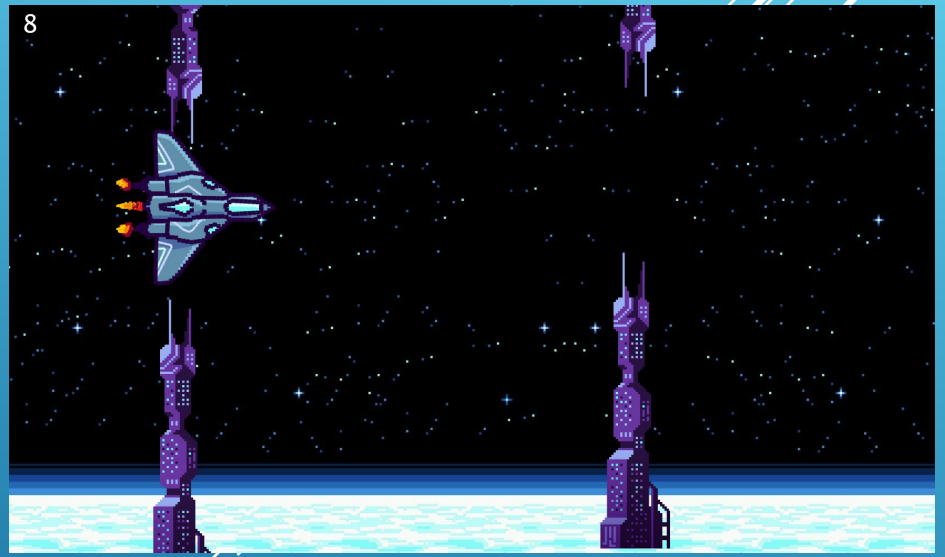
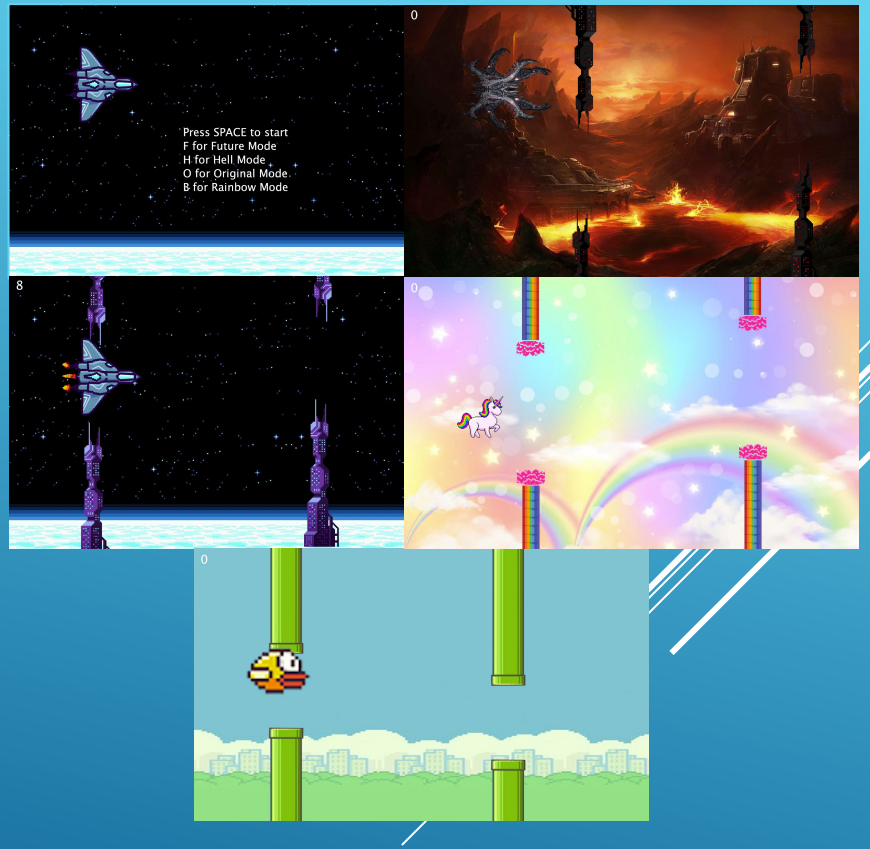
Hover Bird

By: Augustus Rollin and
Yamaan Khundakjie



Game Description

- Similar to Flappy Bird, but instead of flapping their wings, the bird just hovers up and down with arrow keys
- Additional features
  - Different themes, which includes a different character, background, and obstacles
  - A boost function that speeds up the character, with a certain amount of fuel that can be used
  - Up and down arrow keys for each direction



How we Collaborated

- Purpose
  - The reason why we used GitHub was to have a way to work on the code at the same time, and so we could constantly see updates each other were making
- How it works
  - There are 3 steps in order to upload the changes to the web page
    - Stage all files that are changed
    - Commit all the staged changes locally
    - Push local commits to the server
  - You also have to pull down all the changes once you start coding

How to use the terminal

```
# pulls all the latest code
git pull

# shows status of files locally
git status

# stages all files that are changed
git add .

# commits all the staged changes locally
git commit -m 'Message'

# locally commits are pushed to the server
git push

# creates a branch
git checkout -b BranchName

# switch branch
git checkout BranchName
```

New Java Framework Items

- Key Listener
- Point class
- Dimension class
- Config.properties
- IO class
- JFrame
- JPanel
- try{}catch(){}
- Sound class
- AudioInputStream class
- Clip class
- AudioSystem class
- FloatControl class
- Hashmaps

Timeline

Week 1 May 8 - May 14

Monday - Create a plan/idea on what to work on for the project
Tuesday - Work on the class hierarchy
Wednesday - Work on the UML Diagram to go with the class hierarchy
Thursday - Work on the UML Diagram to go with the class hierarchy
Friday - Start working on the pseudocode for some of the classes that are required for the game to function

Week 2 May 15 - May 21

Monday - Start working on some of the code for some of GameRunner/Characters/Obstacles
Tuesday - Work on those classes some more to setup a nice template
Wednesday - Setup keylistener so the character can go up/down
Thursday - Start the ReadMe
Friday - Start the rest of the random classes

Week 3 May 22 - May 28

Monday - Work on sounds
Tuesday - Work on more sounds
Wednesday - Setup config file for ease of access/modularity
Thursday - Resize the window to make it full screen
Friday - Fix the errors with the pipes

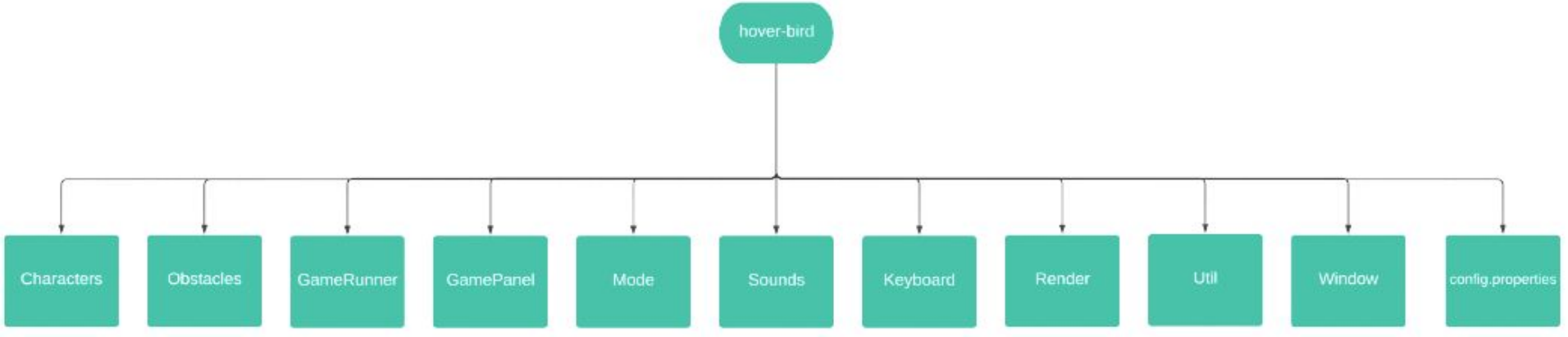
Week 4 May 29 - June 4

Monday - Work on creating the Mode class
Tuesday - Add new images for gamemodes
Wednesday - get the gamemodes to fully work
Thursday - Set up variables for boost
Friday - Work on boost and new modes

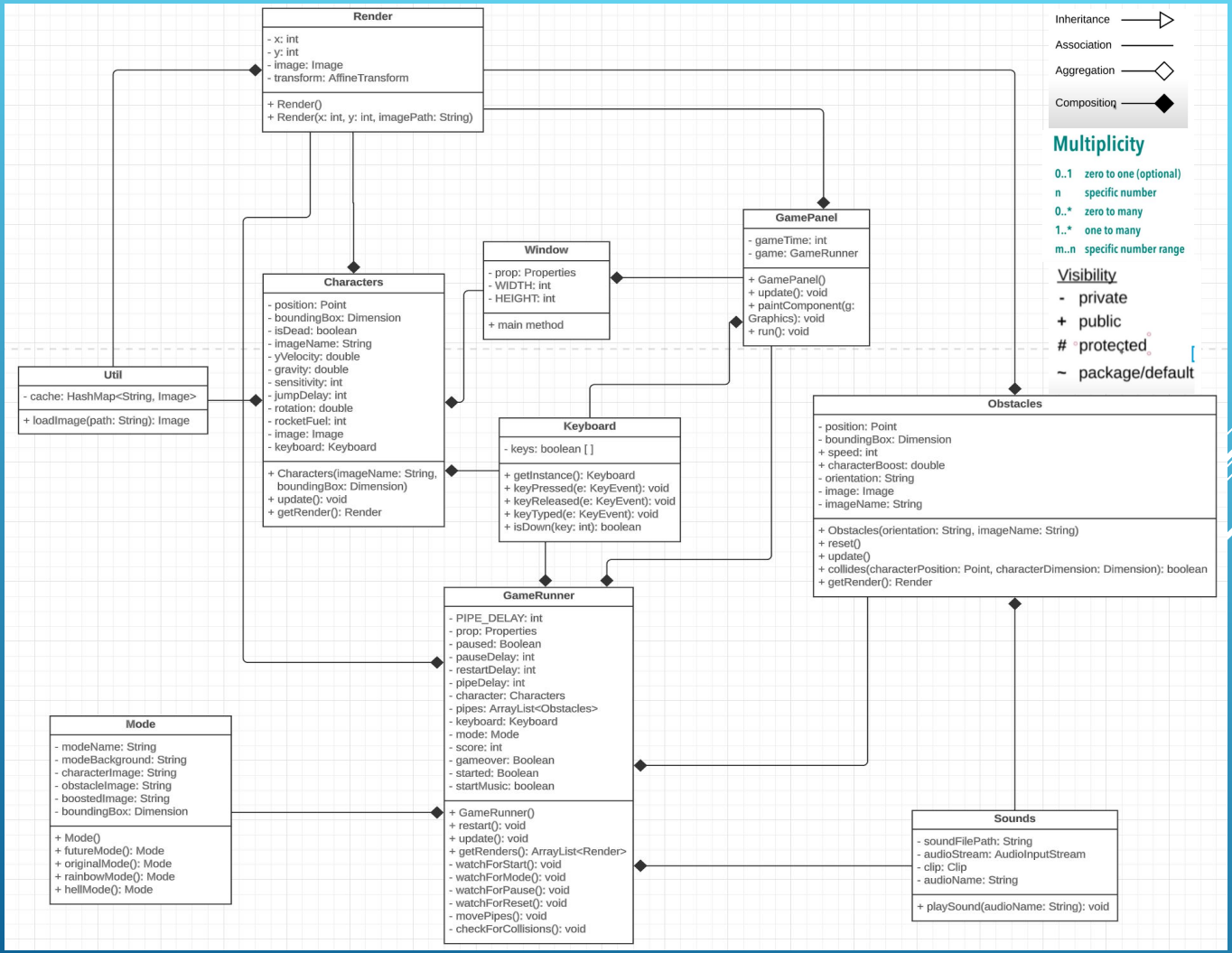
Week 5 June 5 - June 7

Monday - Finish up some minor details in the code/work on presentation
Tuesday - Finish up some minor details in the code/work on presentation
Wednesday - Finish up some minor details in the code/work on presentation

Class hierarchy



UML Diagram


Render
- x: int
- y: int
- image: Image
- transform: AffineTransform
+ Render()
+ Render(x: int, y: int, imagePath: String)
Inheritance
Association
Aggregation
Composition
Multiplicity
0..1 zero to one (optional)
n specific number
0..* zero to many
1..* one to many
m..n specific number range
Visibility
- private
+ public
# protected
~ package/default
GamePanel
- gameTime: int
- game: GameRunner
+ GamePanel()
+ update(): void
+ paintComponent(g: Graphics): void
+ run(): void
Window
- prop: Properties
- WIDTH: int
- HEIGHT: int
+ main method
Characters
- position: Point
- boundingBox: Dimension
- isDead: boolean
- imageName: String
- yVelocity: double
- gravity: double
- sensitivity: int
- jumpDelay: int
- rotation: double
- rocketFuel: int
- image: Image
- keyboard: Keyboard
+ Characters(imageName: String, boundingBox: Dimension)
+ update(): void
+ getRender(): Render
Util
- cache: HashMap<String, Image>
+ loadImage(path: String): Image
Keyboard
- keys: boolean [ ]
+ getInstance(): Keyboard
+ keyPressed(e: KeyEvent): void
+ keyReleased(e: KeyEvent): void
+ keyTyped(e: KeyEvent): void
+ isDown(key: int): boolean
Obstacles
- position: Point
- boundingBox: Dimension
+ speed: int
+ characterBoost: double
- orientation: String
- image: Image
- imageName: String
+ Obstacles(orientation: String, imageName: String)
+ reset()
+ update()
+ collides(characterPosition: Point, characterDimension: Dimension): boolean
+ getRender(): Render
GameRunner
- PIPE_DELAY: int
- prop: Properties
- paused: Boolean
- pauseDelay: int
- restartDelay: int
- pipeDelay: int
- character: Characters
- pipes: ArrayList<Obstacles>
- keyboard: Keyboard
- mode: Mode
- score: int
- gameover: Boolean
- started: Boolean
- startMusic: boolean
+ GameRunner()
+ restart(): void
+ update(): void
+ getRenders(): ArrayList<Render>
- watchForStart(): void
- watchForMode(): void
- watchForPause(): void
- watchForReset(): void
- movePipes(): void
- checkForCollisions(): void
Mode
- modeName: String
- modeBackground: String
- characterImage: String
- obstacleImage: String
- boostedImage: String
- boundingBox: Dimension
+ Mode()
+ futureMode(): Mode
+ originalMode(): Mode
+ rainbowMode(): Mode
+ hellMode(): Mode
Sounds
- soundFilePath: String
- audioStream: AudioInputStream
- clip: Clip
- audioName: String
+ playSound(audioName: String): void

Basics to the Code/Design Process

- A class named Characters is used as a blueprint for each character in the game modes, such as the spaceship, the hell ship, and the bird.
- Characters holds methods and attributes for the game characters that determine properties such as size, image name, x, y, etc. Methods are used to Render the characters, update the screen, and listen for user inputs.
- A class named GamePanel is used to run the game on a Single thread, at about 60 frames per second.
- GamePanel inherits JPanel, and uses the paintComponent method from JPanel to render text and images.
- The main method is in the Window class, which initializes a JFrame for the game and adds KeyListener to it.

Basics to the Code/Design Process (cont)

- The Obstacles class is similar to the Characters class, but the main difference is that the GameRunner class will use an ArrayList of Obstacles. It also contains code to detect for collisions.
- GameRunner contains many important methods used for pausing, updating, moving pipes, checking for collisions, and basically just running the game.
- The Mode class contains many different properties for each game mode.
- The Sounds class contains methods that are used to control audio in the game.
- The Util class contains a hashmap and the code needed to render images onto the JFrame.
- The Render class is used to render and transform images at a certain x and y value.

Live Demo/Game Test

Roadblocks

1. Getting the hitboxes to work properly for collisions
2. Getting the Obstacles to spawn in the right place
3. Making the score work properly (for passing the pole at normal speed and the boost)
4. Issues with merging
5. Resizing panel to include the resizing of the images
6. Frames per second
7. Getting config.properties file to work


```
private void checkForCollisions() {

    for (Obstacles pipe : pipes) {
        Sounds music = new Sounds();
        if (pipe.collides(character.position, character.boundingBox)) {
            gameover = true;
            character.isDead = true;
            Characters.rocketFuel = 10;
            Obstacles.characterBoost = 0;
        } else if ((pipe.position.x == character.position.x)
                && (pipe.orientation.equalsIgnoreCase(anotherString:"south"))) {
            score++;
        }
    }
```

```
private void checkForCollisions() {

    for (Obstacles pipe : pipes) {
        int buffer = (Obstacles.speed + (int) Obstacles.characterBoost) / 2;
        Sounds music = new Sounds();
        if (pipe.collides(character.position, character.boundingBox)) {
            gameover = true;
            character.isDead = true;
            Characters.rocketFuel = 10;
            Obstacles.characterBoost = 0;
        } else if (((pipe.position.x - buffer <= character.position.x)
                && (pipe.position.x + buffer >= character.position.x))
                && pipe.orientation.equalsIgnoreCase(anotherString:"south")) {
            score++;
        } else {
            System.out.println(pipe.position.x + " " + character.position.x);
        }
    }
```

Future Changes

1. Use a different way to detect for collisions
2. Have a way to resize the window and make the code more modular
3. Add an option for multiplayer
4. Add special effects at certain (Screen zoom in for example)
5. Add a death animation (like an explosion)
6. Add a working GUI to select modes
7. Customize the modularity of the themes (make some modes faster, different fonts, etc...)
8. Add volume settings, game settings, etc…
9. Make the game accessible to mobile devices/touchscreen works
10. Add a high score
11. Add a boost for other characters


```
public Mode() {
    modeName = "";
    modeBackground = "";
    characterImage = "";
    obstacleImage = "";
    boundingBox = new Dimension(width:0, height:0);
}
```

Work Cited

https://depositphotos.com/vector-images/unicorn-pixel.html
https://www.vecteezy.com/free-vector/rainbow-background
https://www.istockphoto.com/vector/big-shiny-glittering-pink-cloud-pixel-art-icon-isolated-on-white-background-dreamy-gm1267527002-371933054
https://www.pngegg.com/en/png-zubnb
https://www.nicepng.com/ourpic/u2q8w7e6e6y3o0q8_flappy-bird-pipes-png-bottle/
https://www.pxfuel.com/en/query?q=flappy+bird+backgrounds4
https://www.pngwing.com/en/search?q=spaceship