Distinction Task 1 – Custom Program UML Class Diagram

Introduction

For Distinction task 1, I have decided to make a game that conforms to the four principles of object-oriented programming. I have named the game "Flappy Bird."

In this game, the player's object "FlappyBird" will be controlled using the Space Key. The bird will be able to move along the vertical axis. A series of poles will appear on the screen positioned randomly as obstacles. The bird must pass through the pole by overcoming the obstacles. If the bird hits the obstacle, the game will come to an end. An 'End screen' will appear on the screen with the current score and the highest score achieved by the player. On another click of the Enter keypad, the game will move to the 'Starting screen' to restart the game.

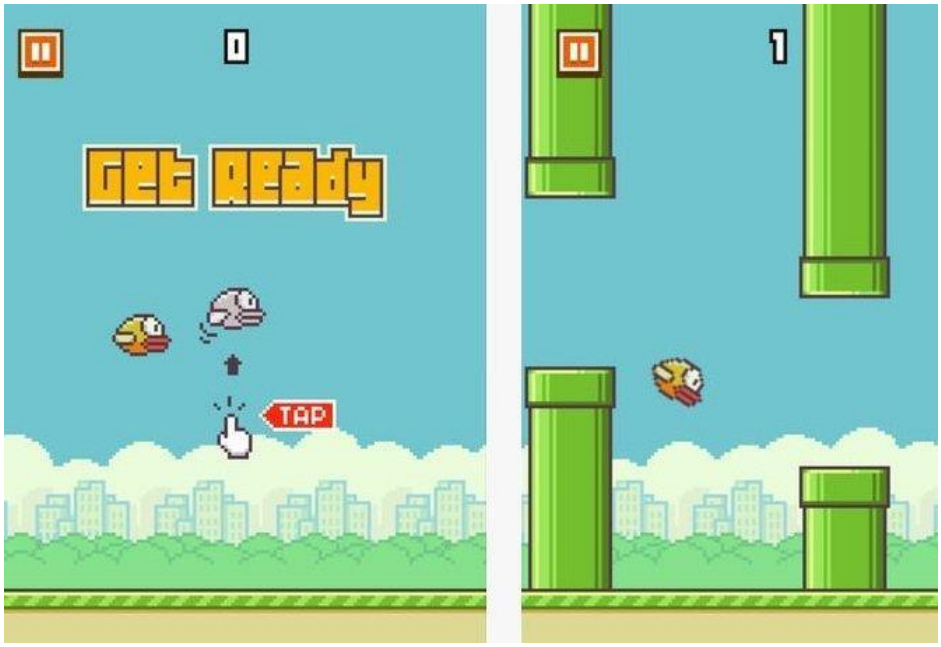


Figure 1: A sample image of the game.

When the bird passes through a pole, the current score will increase by 1. The highest score recorded by the player is saved and displayed at the end of the game. Appropriate sounds are played upon each action of the player.

Another enhancement included in the game is that the bird will alternate between pink and yellow. If the current score is an even number, the color of the bird will be pink. Else, the color will be yellow.

The speed of the bird and the bounce rate of the bird is different for the pink bird and the yellow bird. This is accomplished using polymorphism.

Roles and responsibilities

Classes

Screen: The screen consists of the background screen which sets the view, and the foreground screen consists of two parts. The top of the foreground consists of thin stripe moving in the backward direction, and the bottom of the foreground consists of the floor moving in the forward direction which creates the illusion of the screen moving ahead. This screen also consists of the introduction screen which appears in the beginning of the screen, and an exit screen which appears at the final when the game end. A flappy bird object is introduced in this class which is a part of the check collision function, which checks if the bird has exceeded

beyond the dimensions of the scree; if yes, the function returns a true value and the game ends. The resources are loaded and initialised in the constructor which are necessary for this class.

Pipe: In this class various functions are called. This class is responsible for forming pipes as obstacles to be displayed on the screen. The function CollidedWith() checks if the bird has collided with the pipes. For this purpose, a flappy bird object is introduced in the class which helps determine the status of the collision. Another method called PipePass() helps determine if the bird has successfully passed through the pipe. If true, then the function returns a true value which leads to an increment in the current points. The pipes are constantly moving towards the left at a steady pace and the draw function displays them on the screen. To display them, the LoadResources() function must be initialised in the constructor to load the necessary resources for the class object.

Pipes: This class is a collection class. It contains a list of Pipe. It is responsible for storing the Pipe in it as a list and displays the list on the window by calling the Draw function. It also checks the collision of the bird and the pipe pass function for each Pipe in the list.

Flappy Bird: This is an abstract class with two child classes. This class bounces the bird object to a certain distance along the y-axis and governs the speed of the bird along the x-axis. When the user presses the space bar while the game state is true, the bird bounces along the y-axis. The movement of the bird along the x-axis continues until it hits the "pipe" obstacle. The virtual method Move() handles the animation of the bird displayed on the screen. Polymorphism is also implemented in this class for a method which controls the speed of the bird along the y-axis, and in another method which controls the bounce of a method i.e., how high the bird will bounce per click. The two methods, Bounce() and Speed() return different bounce values and speed values for the two birds. The resources are loaded and initialised in the constructor which are necessary for this class.

- **Red bird:** This class is a child class of the parent Flappy Bird class. This class implements the abstract method by spawning the red bird when the current score of the user is an even number. This class also implements polymorphism by overwriting the abstract method of speeding up the bird along the y-axis every time a red bird is spawned.
- **Yellow bird:** This class is a child class of the parent Flappy Bird class. This class implements the abstract method by spawning the yellow bird when the current score of the user is an odd number. This class also implements polymorphism by overwriting the abstract method which controls how high the bird will bounce along the y-axis. It reduces the bounce amount of the bird.

Scores: The current scores are displayed in the corner of the screen when the game state is true and then at the end once the game ends. The highest scores are updated if the user's current scores are more than the previously recorded highest scores. The highest scores are stored in a file and can be fetched and updated at any time. The current scores are updated by an amount of 1 if the bird successfully passes through the pole without collision. This Update_s() method is called when the PipePass() method returns a value of true. It also displays the instructions to play the game on the window. The resources are loaded and initialised in the constructor which are necessary for this class.

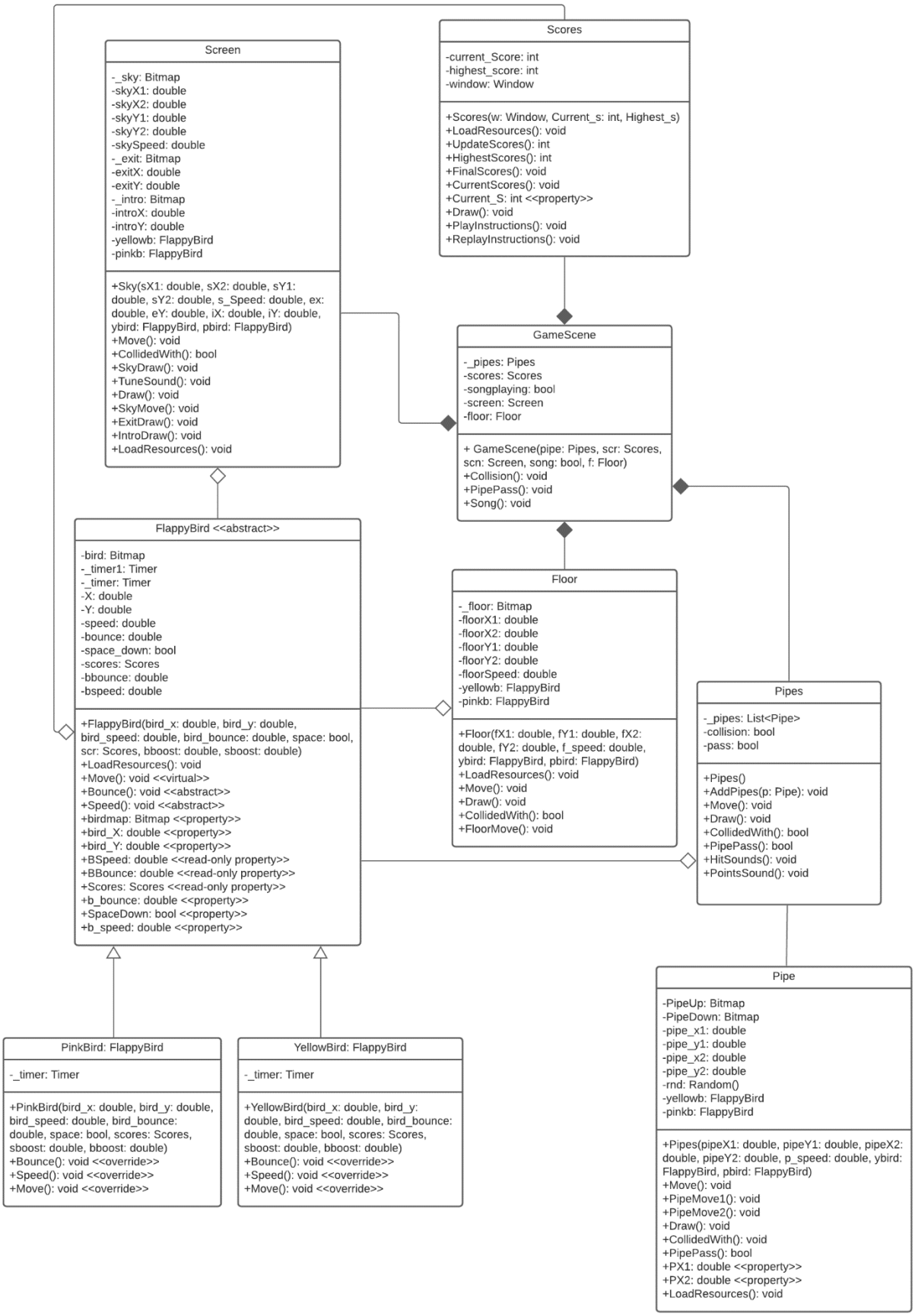
Floor: In this class, the floor object is constructed which moves towards the left at the same speed as the pipes which gives the illusion that the bird is moving ahead wherein reality it is static. This resources are loaded and initialised in the constructor which are necessary for this class. A CollidedWIth() method is implemented in this class which checks if the bird has collided with the floor. It is a Boolean function which returns true if the bird has collided with the floor. For this purpose, a flappy bird object is made a part of the class.

GameScene: This class is composed of other classes which come together to combine a common function. In this class, a function is constructed which plays the background music of the game on a loop. Another function called Collision combines the collision methods of the Pipes, Floor, and Screen class. It also has a method derived from the PipePass function in Pipes class.

Main program: In the main program, there are three states of the game which are managed using Boolean values. The states are game_on and gameover. For the introduction state, the bird is static along the x and y axis and animation is carred out using the Move() method. Instructions are displayed on the screen to start the game. This can be done by pressing the space key. A wing flapping sound is produced each time the user presses the space key.

Once the user presses the space key, the Boolean value on game_on changes to true and the game begins. Obstacles appear on the screen from the right in random positions and the player can now control the movement of the bird along the y-axis using the Space key. If the value of the PipePass function is true, then the points are updated by one each time the bird passes through the pipe obstacle and a point is gained in the current scores. The current scores are displayed in the top left corner of the screen. A points sound is made each time a point is collected. If the bird collides with the floor, the pipes, or extends beyond the dimensions of the screen, a hit sound is produced and then the state of the Boolean gameover becomes true and the game on Boolean becomes false.

When the state of gameover Boolean is true, and game_on Boolean becomes false, then the exit screen is displayed containing the scores gained, the highest scores yet achieved, and instructions to replay the game.


Scores
-current_Score: int
-highest_score: int
-window: Window
+Scores(w: Window, Current_s: int, Highest_s)
+LoadResources(): void
+UpdateScores(): int
+HighestScores(): int
+FinalScores(): void
+CurrentScores(): void
+Current_S: int <<property>>
+Draw(): void
+PlayInstructions(): void
+ReplayInstructions(): void
Screen
-_sky: Bitmap
-skyX1: double
-skyX2: double
-skyY1: double
-skyY2: double
-skySpeed: double
-_exit: Bitmap
-exitX: double
-exitY: double
-_intro: Bitmap
-introX: double
-introY: double
-yellowb: FlappyBird
-pinkb: FlappyBird
+Sky(sX1: double, sX2: double, sY1: double, sY2: double, s_Speed: double, ex: double, eY: double, iX: double, iY: double, ybird: FlappyBird, pbird: FlappyBird)
+Move(): void
+CollidedWith(): bool
+SkyDraw(): void
+TuneSound(): void
+Draw(): void
+SkyMove(): void
+ExitDraw(): void
+IntroDraw(): void
+LoadResources(): void
GameScene
-_pipes: Pipes
-scores: Scores
-songplaying: bool
-screen: Screen
-floor: Floor
+ GameScene(pipe: Pipes, scr: Scores, scn: Screen, song: bool, f: Floor)
+Collision(): void
+PipePass(): void
+Song(): void
FlappyBird <<abstract>>
-bird: Bitmap
-_timer1: Timer
-_timer: Timer
-X: double
-Y: double
-speed: double
-bounce: double
-space_down: bool
-scores: Scores
-bbounce: double
-bspeed: double
+FlappyBird(bird_x: double, bird_y: double, bird_speed: double, bird_bounce: double, space: bool, scr: Scores, bboost: double, sboost: double)
+LoadResources(): void
+Move(): void <<virtual>>
+Bounce(): void <<abstract>>
+Speed(): void <<abstract>>
+birdmap: Bitmap <<property>>
+bird_X: double <<property>>
+bird_Y: double <<property>>
+BSpeed: double <<read-only property>>
+BBounce: double <<read-only property>>
+Scores: Scores <<read-only property>>
+b_bounce: double <<property>>
+SpaceDown: bool <<property>>
+b_speed: double <<property>>
Floor
-_floor: Bitmap
-floorX1: double
-floorX2: double
-floorY1: double
-floorY2: double
-floorSpeed: double
-yellowb: FlappyBird
-pinkb: FlappyBird
+Floor(fX1: double, fY1: double, fX2: double, fY2: double, f_speed: double, ybird: FlappyBird, pbird: FlappyBird)
+LoadResources(): void
+Move(): void
+Draw(): void
+CollidedWith(): bool
+FloorMove(): void
Pipes
-_pipes: List<Pipe>
-collision: bool
-pass: bool
+Pipes()
+AddPipes(p: Pipe): void
+Move(): void
+Draw(): void
+CollidedWith(): bool
+PipePass(): bool
+HitSounds(): void
+PointsSound(): void
Pipe
-PipeUp: Bitmap
-PipeDown: Bitmap
-pipe_x1: double
-pipe_y1: double
-pipe_x2: double
-pipe_y2: double
-rnd: Random()
-yellowb: FlappyBird
-pinkb: FlappyBird
+Pipes(pipeX1: double, pipeY1: double, pipeX2: double, pipeY2: double, p_speed: double, ybird: FlappyBird, pbird: FlappyBird)
+Move(): void
+PipeMove1(): void
+PipeMove2(): void
+Draw(): void
+CollidedWith(): void
+PipePass(): bool
+PX1: double <<property>>
+PX2: double <<property>>
+LoadResources(): void
PinkBird: FlappyBird
-_timer: Timer
+PinkBird(bird_x: double, bird_y: double, bird_speed: double, bird_bounce: double, space: bool, scores: Scores, sboost: double, bboost: double)
+Bounce(): void <<override>>
+Speed(): void <<override>>
+Move(): void <<override>>
YellowBird: FlappyBird
-_timer: Timer
+YellowBird(bird_x: double, bird_y: double, bird_speed: double, bird_bounce: double, space: bool, scores: Scores, sboost: double, bboost: double)
+Bounce(): void <<override>>
+Speed(): void <<override>>
+Move(): void <<override>>