**Platformer Game**

**IMPORTS**

Using the Java library JavaFX, I was able to create an application that runs on my MacBook.

Used “import javafx.scene.shape.Rectangle” so that I could create my rectangle background of my application.

Used “import javafx.scene.layout.Pane” so that I could create the two parts of my application, the application itself and the game

Used “import javafx.animation.AnimationTimer” so that I could determine flow of game and control the movement/momentum of the player (box) by applying transitions to it

Used “import javafx.application.Application” to create the whole java application. “import…node” used to create model objects such as the player and platforms. “import…stage” used to contain all of the objects of java application.

Used “import java.util.HashMap” and “import…KeyCode” to include all map operations so that I could initialise keys to be pressed so that user can interact with the application.

Used “import…color” so that I could set the colour of the different parts of the application.

Used ”import…Point2D” so that I could set the coordinates of x and y.

**GAMECONTENT**

**“public static void main…”** used to launch the application

**“Pane ApplicationRoot”** used to create whole application containing other nodes/roots such as “GamesRoot” which include all game entities such as the player and platforms.

**“HashMap<KeyCode…”** used to check if a key is pressed on keyboard which returns true or false hence boolean.

**“int WidthofLevel”** to set size of level.

**“ArrayList<Node>…”** used to set/store/use different entities/nodes such as the platform.

**“Node player”** used to create the player (box) itself.

**“Point2D playerVelocity”** used to store player speed/location using x and y coordinates.

**“Boolean canJump”** used to enable player to jump over gaps between platforms.

**“void initContent”** to create background, player, platforms, size of level, follow player along platforms and to add all content to the application:

* “Rectangle bg = new ….” Create a white background with said dimensions.
* Create size of level determined from the CreateLevel class.
* **“for()…”** loop will go through each element of CreateLevel class and using it will create platforms: - “String line…” take each row one by one and put through loop. – “Switch(line.charAt)” to take each character from each row and apply cases 0 or 1. – case 0 nothing happens, case 1 green platform created and added within application with following dimensions. – “player = Createentity” player is created with following dimensions ***(scroll down and reference private node CreateEntity(……),***  first two values as x and y coordinates and second two width and height of platform and player and finally colour of each – “player.translateXproperty…” used to follow player as they move along, ***explain*** – “ApplicationRoot.getchildren…” used to add background and and other game entities to application window.
* **“Private Node CreateEntity…”** used to create entities background, player and platform: - “Rectangle entity..” to create application window itself – following used to set x and y coordinates of player and colour – “GamesRoot…” adding entities to actual game withon the application.
* **“override: public void start(Stage…)”** to create application, calls initContent metod from above to create everything: - “Scene scene = …” Creates application window with all application roots inside. – “scene.setonkeypressed..” gets code from event to see what key is pressed and what key is released. – “primaryStage.setTitle..” sets title of application window. – finally following shows the application itself containing the scenes.
* **“pivate void update()”** This is where the game logic happens: player jumps and moves, - “if (isPressed(KeyCode.UP)…” To make player jump, here we see if key is pressed and if it is player is moved up the y-axis thus jumps, - “if (isPressed(KeyCode.LEFT)” this if statement is to detect if the left arrow key is pressed and to move the player to the left, here again 5 represents how fast the player moves, -“if (isPressed(KeyCode.RIGHT)” if statement here detects if right arrow key is pressed and if it is it moves the player to the right, - “if (playerVelocity.getY() < 10)” this if statement calculates the location and speed of the player and controls how high the player jumps, - The next bit then makes the player jump a certain bit and how quick the jump is
* **“private Boolean isPressed…”** Getting keycode form defined map to see if key is pressed or not, If not pressed returns "false" by default, If it is pressed it will return the boolean value of that key and therefore carry out the task that key is supposed to, such as move left or right.
* **“private void jumpPlayer…”** if player is told to jump then, - “player velocity…” y-cordinate is opposite in java effects screen, thus the minus value for 35 makes the player jump a certain distance at a certain speed – next bit disables the player to double jump
* **“private void moveplayerX..”** this class moves the player on the x-axis and does collision detection between the player and the platforms, - “Boolean movingRight..” moves player right with the value of 5 from above update method, - The “for loop” perfroms the following regarding each platform the player touches/intersects with, - “if (player…)” gets bound/location of player and sees if it intersects with the bound/location of a platform, if yes a collision is detected – “if (movingRight)… this if statement checks if right side of player is touching if yes.. – “if (player.get…)” this statement if the player and platform is in the same x position so collision is detected and player stops moving – next bit is moving left and it does exactly the same as above – “player.setTrasnlateX…” if no collision this statement runs and this moves the player right by one unit or left by minus one unit this is what the (movingRight ? 1 : -1) statement does the question mark represents an if-else statement. The values 40 and 60 represent width of the player and the platforms.
* **“movePlayerY”** This method moves the player according to the y-axis. It is exactly the same as the “movePlayerX” method above. The only difference is “player.setTranslateY(player.getTranslateY() - 1); canJump = true;” which basically prevents player from constantly colliding with the platforms in other words it allows the player to keep moving up and down the y-axis (prevents it from sticking to one position).

**GitHub URL for Repository**

**https://github.com/Khurram1997/DT211C-2**

**YouTube Video URL**

**https://youtu.be/92bAgQzSSCk**