Team Name: Sparta

Team Members: Alex Portolese and Sam Stein

Responsibilities:

GUI and Level Design: Sam

Player animations and object collision: Alex

Lessons learned:

Some of the lessons we learned from this project is how to effectively handle multiple dynamic objects with vectors and handling object collision through QRect objects. We also learned how to handle multiple key presses, such as moving left or right while jumping. We also learned how to make objects move while a key is held down without any input lag from the keyboard’s poll rate through the use of KeyPressEvent and KeyReleaseEvent and make the objects stop moving when no key is being pressed at all.

Code Explanation:

In donut.cpp:

1. **drawDonut(QPainter &painter)** : draws the donut object on screen.
2. **changeImage(QString str)** : changes the existing image with the new one defined in a QString object.

In enemy.cpp:

1. **setDirection(int direc)** : takes in an int value and compares it to a set of enumerations through the use of a switch statement. If the int value matches an enumeration, then the objects image will change so it faces the opposite direction.

In highscores.cpp:

1. **HighScores(QWidget \*parent)** : opens up the highscores.txt file and reads in the values to be displayed on the highscores.ui object.
2. **on\_reset\_clicked()** : Opens up the highscores.txt file through the use of an ofstream and replaces all existing values with zeros and then refreshes the values displayed in the highscores.ui by using an ifstream.

In levelbase.cpp:

1. **drawLevelbase(QPainter &painter)** : defines a QRect object for the floor and a QBrush object for the floor’s texture. The painter then draws the rect and fills it with the texture.

In mainWindow.cpp:

1. **MainWindow(QWidget \*parent)** : initializes the mainWindow UI as well as initializes the object for the controls, contact us, highscores, and superCopGame windows. It also displays a background to the mainWindow.
2. **On\_actionExit\_triggered()** : Exits the program
3. **On\_actionControls\_triggered() :** checks if the controls window has been opened before prior to opening a new one. Displays the game controls window.
4. **On\_actionContactUs\_triggered()** : checks if the contactUs window has been opened before prior to opening a new one. Displays information about the developers
5. **On\_actionHighScores\_triggered()** : checks if the highscores window has been opened before prior to opening a new one. Displays the players 5 highest scores per setting.
6. **On\_Easy\_1P\_triggered(),** **on\_Medium\_1P\_triggered(), on\_Hard\_1P\_triggered()** : Checks if the superCopGame window has been opened before prior to opening a new one. Displays the Super Cop game on the desired difficulty.

In platform.cpp:

1. **drawPlatform()** : creates a QRect object for the platform and a QBrush object for the texture. The painter then draws the rect object and fills the rect with the texture.
2. **setActive(), isActive()** : Accessor and Mutator methods for the active state Boolean variable. This is for the dynamic objects stored in vectors in the superCopGame class.

In player.cpp:

1. **Player(QWidget \*parent)** : initializes all of the player object variables such as the player texture, image dimensions, player position as well as player collision flags and free movement borders.
2. **playerScreenPos()** : Check where player is on screen. If within a predefined rect, do not scroll screen. If on edge of rect, move camera in direction player is running.
3. **playerAction(int action)** : Takes in an int value that is passed from the superCopGame class file. This function checks if the current key pressed (action) is the same as the last key pressed (lastActionPressed). Within that statement, another if statement is checking if the up key was pressed (upPressed, controlled by the jump function). If the up key is not pressed, set the animation frame counter to zero. If up is pressed this code block is skipped over and the last key pressed becomes equal to the action passed. Once this code block has been executed, the function moves on to a switch statement where the passed action parameter is compared to one of five predefined outcomes. Depending on which outcome evaluates to true, the player will execute a corresponding function.
4. **jump()** : this method is called when the action evaluated in playerAction is equal to UP (int value of 2). It first updates the frame counter by one, sets the player jump flag to true, and the upPressed flag to true; before heading into an if statement. This statement checks if the current frame value is greater than zero and less than 10 then the QString imagePath is defined. A second if statement then checks if the current frame is greater than zero and less than five, if this evaluates to true, then the player ascend flag is set to true before executing a switch statement. This switch statement checks what the current player direction is (WEST = -1, EAST = 1). Depending on which on evaluates true, the player’s image will change so that he is facing the correct direction. The player will then ascend 30 pixels up. If the frame count is greater than five, then the players ascend flag is set to false before executing a switch statement. This switch statement once again checks the player direction and sets the correct image. The players position is not changed in the second switch statement of the jump function. If the frame count is greater than 10, the player enters a standby state.
5. **roll() :** this function controls the players roll animation. Similar to the jump function, the frame counter is incremented by one then goes to an if statement. This if statement checks if the frame counter is greater than o and less than 9. If it is true, then a QString object is declared and the players rolling flag is set to true. After this, another if statement is executed to check if the frame counter is greater than zero and less than five. If this is true then a switch statement is executed to check which direction the player is facing. Once a direction has been determined the function checks if the roll is going to move the player past the free movement bounds. If the player will not roll past the bounds, then the player will roll forward so many pixels per frame determined by the player speed set by the game difficulty. If the player will roll past the bounds, the roll animation continues but the player position does not change. Once the frame counter has gone above 5 frames, the function once again checks the players direction and if they will collided with the free movement bounds. If it is determined they will not roll past the bounds, the player rolls fewer pixels per frame than before using a similar calculation as before. Once the frame counter exceeds 9 frames, the rolling flag is set to false and the player enters the standby state.
6. **run() :** this function first checks the player flags to see if he is in the air, and if the ground collision flag, the platform collision flag, and the wall collision flag are all false. If this is the case, then the function will check to see if the player is colliding with the right bound. If this evaluates to false then the player’s x position will change by one pixel. If the player is colliding with the bound, then the player’s x position will not change. This is followed by a second if/else statement that checks if the upPressed flag has been set to false. If it has been, then set the frame counter to 5 and run the jump() function. This is to handle the player walking off of a wall or platform object. If the upPressed key is set to true, then the frame count equals the current frame and the upPressed key remains true; the jump() command is then called after this. This else condition is to handle the player moving while jumping. If the first condition determines that the player is on the ground, on a platform, or on a wall, then players jumping and upPressed flags are set to false and the frame counter is incremented by one. A QString object is declared to the location of the next frame in the animation before the function moves on to an if/else statement. This if statement checks if the current frame is greater than zero and less than four. If true, then the image is changed to what is stored in the QString object. It then sets the players movingRight flag to true and sets playerDirection to 1 (EAST/RIGHT). If the statement evaluates to false, then the frame counter is reset to zero, the moveRight flag is set to false and the image is set to the first frame of the run animation.
7. **runInverted() :** this function acts the same as the run function, just in the opposite direction.
8. **standBy() :** this function gets the players current direction and displays the appropriate image. This function is called when every an animation ends or the user is no longer pressing a valid key.

In superCopGame.cpp:

1. **SuperCopGame(QWidget \*parent)** : The constructor of the class initializes the player object, the floor object, the game’s background, and the timers that control the how often the game updates and when the key press gets polled. It also sets all of the key press flags are set to false here and the lastKeyPress, location, and gamescore variables are all initialized to zero.
2. **KeyPressEvent(QKeyEvent \*evt) and KeyReleaseEvent(QKeyEvent \*evt) :** These two functions work together to detect when the user enters an input and keeps that input active to avoid the operating system’s default delay in telling the difference between a single key press and a key held down. The KeyPressEvent has two extra possible events that check for the pause button being pressed. When a key is pressed, the KeyPressEvent sets the corresponding flag to true and when the system detects the key has been released sets it to false. These inputs are then interpreted in the pollKey slot that is triggered every 5ms by the keyTimer timeout signal.
3. **pollKey() :** this function interperets the flipping of key pressed flags to keep the player animation from stalling while the operating system determines the whether it was a single key press or a held key press. The first set of if/else if statements check for which of the flags is true, if none of the flags are true, then the second set of if/else statements nested in to the final else code block, checks, first, if none of the keys are pressed. If this evaluates to true, then a third nested if/else if statement, the function checks If 3(down arrow/roll) was the last key pressed and if the player is rolling, if this is true, then set the last key pressed to 3. The remaining else if statements carry out similar commands as the if statement, checking if the player is jumping, moving left, or moving right. If none of the statements evaluate to true, the the last key pressed will be zero which triggers the standby function in the player class. If any of the keys evaluate to true if the nested if/else code block, then last key pressed will be set to zero. This is just a safety precaution to ensure that the player does not move un intentionally.
4. **obstacleMovement() :** This function’s purpose is to check if the player is moving left or right and has collided with the free movement bounds. If the function determines that the player has collided with the one of the bounds, then the player’s x position will stop and the level will shift in the opposite direction of the player (i.e. player collided with the right boundary so the level starts to shift to the left.)
5. **updateField() :** This function is called when timer’s timeout signal is triggered. This function updates the player with the last key pressed, runs the obstacleMovement function and updates the screen.
6. **paintEvent(QPaintEvent \*e) :** this function contains all of the donut, enemy, wall, and platform, object creation and player collision handling. If the function, each time one of the object control loops reaches a value that matches the player’s location (defined by the right boundary) the object matching the location is drawn on screen. If It is a donut object, then any collision with the player will result in the user’s score increasing by ten. If the object is an enemy, then colliding with the player head- on will result in a game over while a collision with the player landing on the enemy will result in the enemy being killed. If the object is a wall, then a collision with the player will result in the all movement stopping, even if the player is moving right at the right boundary. If the player land on the wall then player can walk on top of it. If the object is a platform object, then a collision with the player landing on it will allow the player to walk on it, any other form of collision will result in the player phasing through the platform. If the player collides with the level ending object, then a message will appear saying “Level Beaten!” and will exit the program upon clicking “OK”. The paintEvent also handles the drawing of the player and floor objects, as well as displaying the game score up in the corner.
7. **setVecs()** :
8. **setHighScores()** :
9. **setMoveSpeed(int spd)** :

In wall.cpp:

1. **Wall(QWidget \*parent)** : The constructor for the class sets up the initial x screen position and sets the wall’s texture.
2. **drawWall(QPainter &painter)** : creates a QRect object for the wall object and a QBrush object for the wall texture. Then the painter draws the rect object and fills the wall rect with the texture.