**The Problem**

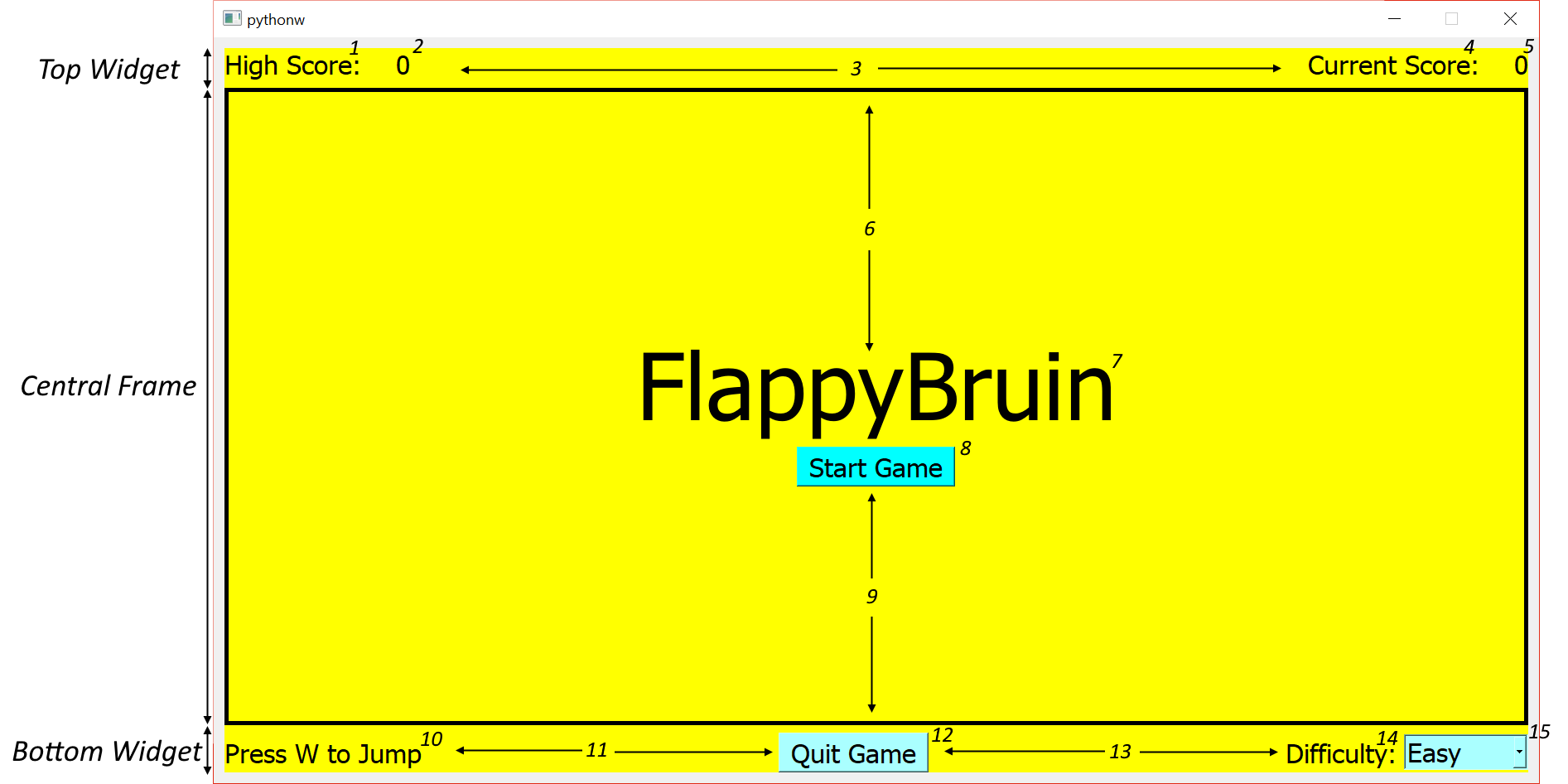
This is a single tap or key press game design. As the player you are Joe Bruin and you must magically flap your way through an endless barrage of dorm buildings, by pressing the W key to flap. Each oncoming dorm has an opening through which Joe can pass. If Joe can adroitly flap through the dorm opening in the building while listening to the Bruin clap, he gets a point. The number of points received for passing each building depends on the difficulty chosen, with higher difficulties earning more points. However, increasing the difficulty also increases the number of floors the buildings have and makes the passage through each building narrower.

**The Solution**

The code for our game is composed of three main files, **flappybruin**.py, **bruin**.py and **building**.py, and supporting files (**lasky**.png, **intro**.mp3, **bruinsfight**.mp3, **upper0**.png, **inner0**.png, **inner1**.png, **lower0**.png, **jump**.ogg, **Joe\_Bruin**.png).

**flappybruin.py**

Defines the overall interface of the game and is responsible for a variety of miscellaneous game-related tasks. We used **QtDesigner** to create the barebones layout of the game.



First, we create a **background widget** with a **vertical layout**. We have three sections, the **top widget** (scores), the **central frame** (game) and **bottom widget** (options). The first frame has a **horizontal layout**, from left to right, **two** **labels (1,2)** a **horizontal spacer (3)** and another **two labels (4,5)**. The center frame has a **vertical layout**, from top to bottom, a **vertical spacer (6)**, a **label (7)**, a **tool button (8)** and a **vertical spacer (9)**. The bottom frame has a **horizontal layout**, from left to right, a **label (10)**, a **horizontal spacer (11)**, a **tool button (12)**, a **horizontal spacer (13)**, a **label (14)** and a **combo box (15)**.

Imports

**sys**: For starting and closing the game safely

**PyQt**: **QtCore**, **QtGui**, **QtWidgets** for the game elements

**bruin**: **Bruin** for the implementation of the bruin player

**building**: **Building** for the implementation of the buildings

**pygame**: Only for sound (because PyQt’s QtMultimedia was glitchy on one of our systems)

FlappyBruin Functions

\_\_**init**\_\_: Initializes the class and all of its member variables, also calls setupUi

- Creates QTimer object used to repeatedly call **animate**

- Creates variables names **player** and **buildings** but sets them to None first

**setupAudio**: Sets up the audio input for the game

- Uses **pygame.mixer** functions to load music and play in a loop

**setupUi**: Sets up all the GUI elements of the game, is called by \_\_init\_\_

- Calls **setFixedSize** to fix the window size and prevent resizing

- Draws a border around the centerframe

- Links the Start Game and Quit Game tool buttons to **gameStart** and **quitgame** respectively

- Links the combo box to **setMode**

**setMode**: Updates the difficulty mode when user interacts with the combo box

- Calls the player and buildlings’ **setDiff**

**gameStart**: Defines actions to start the game

- Stops the menu music, and starts playing the active game music

- Hides the center label and start game button

- Disables the difficulty combo box

- Calls the player and buildings’ **reset**

- Resets current score to 0

**gameEnd**: Defines actions to end the game

- Stops the active game music, and starts playing menu music

- Displays game over with the center label and restart on the start game button

- Reenables the difficulty combo box

**framePress**: Debugging tool used to output to the console the mouse position when clicking on the GUI

**keyPressEvent**: Defines what happens when user presses the ‘W’ key

- Calls player.**jump** when W is pressed

**framePaint**: Constructs a QPainter for the central frame and calls the player and building’s paint functions.

- Instantiates player and buildings when called the first time

- Sets focus on itself when mouse is not hovering over combo box

**eventFilter**: Allows game to switch event focus to the combo box

- Sets focus to the combo box when mouse is hovering over it

**updateScore**: Updates the score and high scores as the game progresses

- If score exceeds high score, update high score as well in real time

**quitgame**: Quits the game safely using **close**

**animate**: Calls update functions for Bruin, Building and the center frame, also calls updateScore

- If player is dead, calls **gameEnd**

- Calls **updateScore** while player is not dead

**Building.py**

Defines the building objects in the game and through them the game difficulty is also set.

We used Adobe Illustrator to draw images representing each section of the building. The building is made up of a combination of images depending on where it is that the player can pass through and the difficulty level of the game. These are illustrated here:



A and B represent the external components of the building, with A being the ground floor and B representing all other floors. A and B are objects that if hit by the player will result in player losing and the game ending. C and D are the internal components of the building with C represent the lobby and D the inside of a dorm room, through which a player can pass to receive points.

Buildings are constructed in three ways depending on the difficulty. Below is an illustration of each case:



A represents the configuration for easy, B for medium and C for hard. Each result is achieved through function in the **Building** class, specifically the **setDiff** function. Depending on whether the **mode** variable is set to easy, medium or hard the **numblocks** variable is respectively set to 3, 4 or 5. This in turn allows the function **paintBuilding** to know how many **floors** to paint, using the **paintBlock** function. **paintBlock** allows us to catch the **gapIndex** variable to draw where the opening for Joe to flap through should be located. **gapIndex** is randomly set so that each building has a randomly set opening. Furthermore, there are always exactly three buildings drawn to the screen at one time. The buildings move from right to left at constant velocity and are redrawn with a new randomly selected gap location when they exit the left hand side of the screen.

Imports

**PyQt**: **QtGui** for game elements, specifically the **QPainter** and **QImage** classes.

**random:** For generating random integers using **randint**.

Building Functions

**\_\_init\_\_:** Initializes the Building class, loading in image files, setting the building count and initial drawing coordinates.

-Uses **QImage** to load png files representing the different building segments

-Uses **reset** to call other function that set initial coordinates

**frame\_width:** Captures the current frame width so that it can be used to adjust object sizes when window size changes.

-uses the **QtGui** function **frame.geometry().width()**

**frame\_height:** Captures the current frame height so that it can be used to adjust object sizes when window size changes.

-uses the **QtGui** function **frame.geometry().height()**

**randomizeGap:** randomly selects an index value within the range of building floors.

-Uses the **randint** function from the random class.

-Also sets values to frame collision coordinates of the upper and lower parts of the building.

**update:** Used to redraw the building when the window size is changed.

**setDiff:** Sets the game difficulty by changing the number of building sections.

**paintBuilding:** paints building in pieces using **paintBlock**.

**paintBlock**: Paints a floor of the building.

-Uses the **drawImage** from the **QPainter** classto draw the image on the frame.

**resize:** Redraws the building coordinates when the window size is changed.

-Uses the **frame\_width** and **frame\_height** class member function.

**reset:** Resets the building class attributes

**Bruin.py**

Constructs the player object. Updates position, velocity, and acceleration as well as success in the game.

The **\_\_init\_\_()** function begins by setting the scale- and score- -Factors that vary by difficulty setting which is established by **setDiff()**. We then set our frame and upper bound variables. Our **reset()** function wil set the position of our player at 1/4th distance from the left of the screen, and in the middle of the height of the screen. With no velocity and an acceleration that is proportional to the height of the frame. This function will also set a variable in Building to determine if the player is inside the column of the building and a dead variable to store whether or not the player has it the edge of a building structures and died. We then move on to our **w()**, **h()**, and **lb()** functions that return the frame width, height, and lower bound respectively. Our **buildingChecks()** functions increases the score by a score Factor when the player exits a building safely. The **jump()** function plays the jumping sound with each associated jump/’W’ press. The **paintBruin()** function binds the image of Joe the Bruin to our character object point. Finally our **update()** function adjusts player position, velocity, and acceleration, as well as checking for collisions.

Imports

**PyQt**: **QtGui** for game elements, specifically the **QPainter** and **QImage** classes.

**Pygame:** for audio capabilities

Bruin Functions

**\_\_init\_\_():** Initializer

Uses QImage to import image of Joe Bruin

Uses pygame to import sound of a jump

**w():** Returns frame width

Uses **frame.geometry().width()** to access frame width

**h():** Returns frame height

Uses **frame.geometry().height()** to access frame width

**lb():** Returns lower bound of playing screen which depends on .ub member variable

**reset():** Resets the player to initial starting attributes. Saves the high Score.

**buildingCheck():** Checks whether or not player is in a building column and then checks whether or

not it has hit the building if it has not hit the building, then it adds a scoreFactor to

the score once it exits the building

**setDiff():** Sets scaleFactor and scoreFactor depending on the difficulty setting

**jump():** Implements the jump sound as it adjusts the verticle velocity to simulate a jump

**update():** Updates the current position, velocity, and checks for collisions. If collision occured,

changes dead to True.

**paintBruin():** Binds the image of Joe the bruin to the character positon