mobile game development assignment

Austin Walkley, Zhang Fan, Callum Rosewarne-Reece

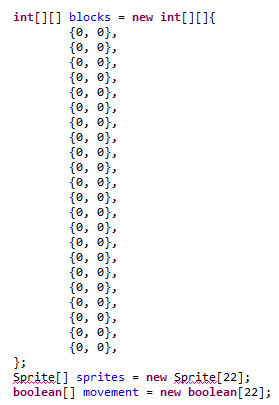
Runner Game

Austin Walkley (WALAN002) Section 1

The sections of code that I wrote were the blocks which the user has to avoid, the touch detection and the collision detection.

# Bad Blocks

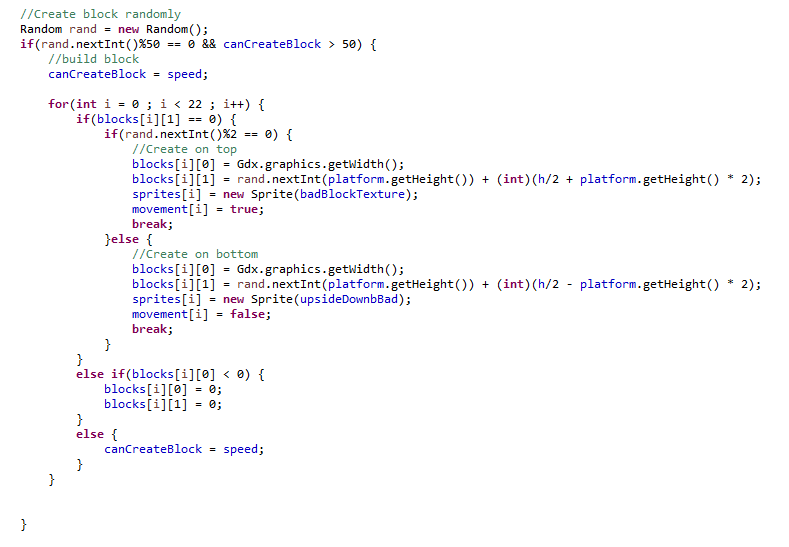
The ‘bad blocks’ as I call them were drawn by myself and were originally designed to move along the bottom dynamically. I later decided that to increase difficulty of the game, I would make them bounce on the platform. I followed a data-oriented design when creating the blocks by creating arrays which would store the position, direction and sprite of these blocks. I needed a sprite array because I would spawn them on both sides of the platform and therefore flip them. Code for the arrays and creation of these blocks can be seen below.

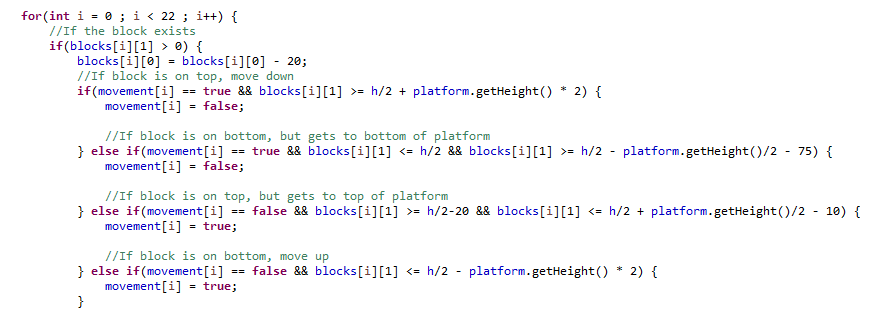


These are the arrays which store the data for the blocks. These arrays are populated by the code shown below. The blocks array is multidimensional and stores the x and y values for the blocks. I decided to make all the arrays size 22 because there will never be more than 22 blocks created on the screen at a time. The movement arrays is of type Boolean because a block can either be moving up or down, so true or false respectively.

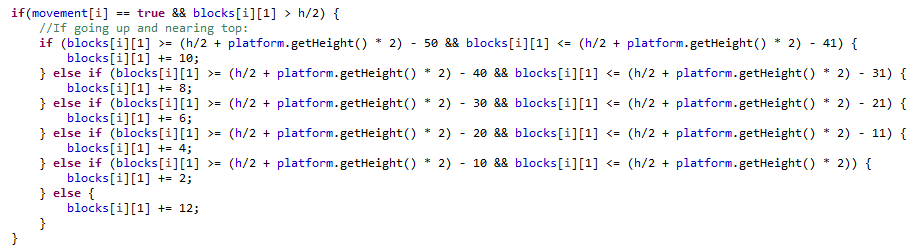
This code populates the arrays with values which are later used to place the blocks on the screen at the desired position. As seen, I use a random to determine whether or not to create a block. By doing this, I can ensure that blocks will not be created at a static rate, adding a little challenge for the user. I also have an integer value “canCreateBlock” which must be greater than 50 for a block to be created. I used this to ensure that blocks aren’t created on top of each other.

Next, I use a random value to determine whether or not to place the blocks on the top or the bottom. I did this for the same reason that I used the random value earlier. Finally, in the else if down the bottom, I check whether the ‘x’ value of the block is below 0 (meaning it’s off the screen). If it is, then I recycle the array space so that another block may occupy it.





In this section of code shown above, I ensure that blocks will bounce at the correct place. I go through my blocks array and I check their ‘y’ values. I check their ‘y’ values because I know that this value will never be 0, and if it is then it will be recycled. I then change the ‘x’ value of the block by 20 pixels to the right to give the impression it is moving towards the player. After this I check the boundary values of the ‘y’ value, and if this value corresponds to the top or bottom of the screen, I will change the direction of it. I also check whether the block is going into the platform. If it is, I change the direction as well.

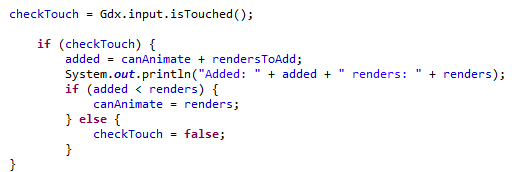


In the code above, I extended on the previous code snippet by creating a bouncing effect. I did this by checking whether or not the block was nearing a boundary. If a block was nearing a boundary, I would check how close it is to the boundary and slow it down by a certain amount based upon how close it is. I had to do this for all four possible directions (Going up and near top, going up and near bottom, going down and near bottom and going down and near top). I only showed one because of how long the code was and I assume you can get the gist of it from one snippet.

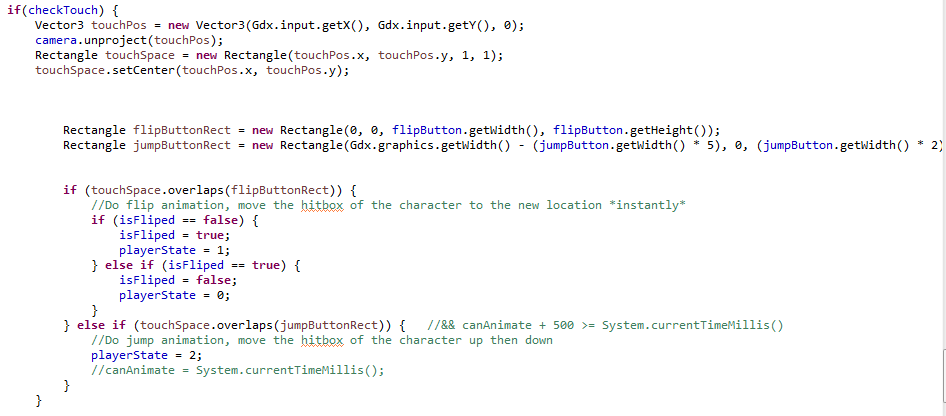


The code snippet above re-draws the blocks based upon their new x and y position, which was updated by all the code snippets above.

# Touch Detection



In this code snippet, I first check whether or not the screen has been touched. If it has, I ensure that it’s been a user defined amount of renders since the last touch (in the case of this assignment, I use 50). If there hasn’t been 50 renders since the last touch, then it will change checkTouch to false and therefore it won’t change the state of the player.



In this section of code, I check whether or not the screen has been touched. If the screen has been touched, I create a vector3 object which is then unprojected by the camera. After doing this, I turn this unprojected location into a rectangle which can be easily compared to a sprite or another rectangle.

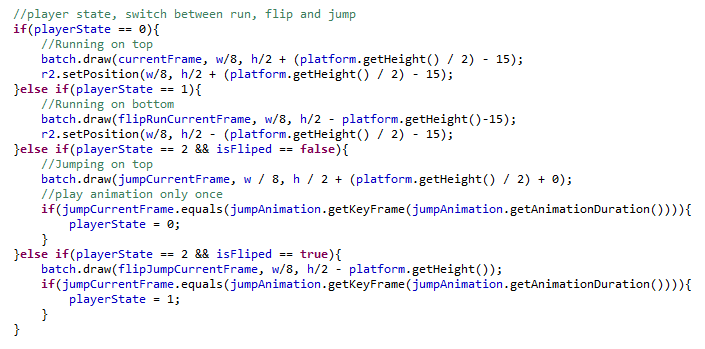
Next I create a rectangle for the flip button and rectangle for the jump button. By doing this, I am able to compare the touch location to the rectangle. Lastly, I check whether these rectangles overlap, and if they do then the player will do the animation which corresponds to the sprite clicked.

# Collision Detection

Finally, I was responsible for collision detection. In order to do all my collision detection, I used rectangles. Rectangles are very easy to use for collision detection because you need only check whether or not they overlap.



The rectangle created above is responsible for the rectangle which bounds the player.

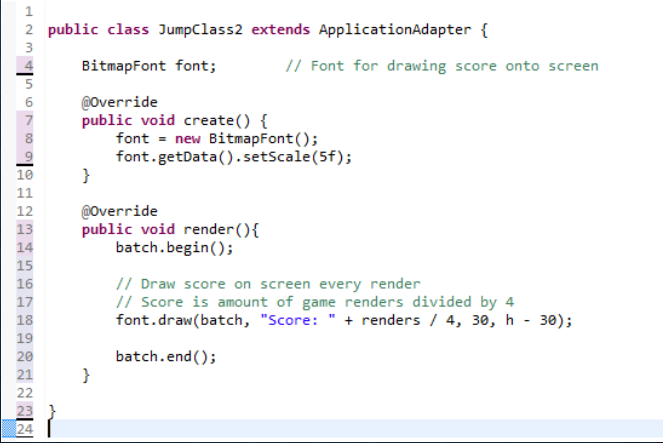


The code snippet above was mostly done by Toby, but I was responsible for the hitbox of players, so I made it so that when you press flip, the players’ hitbox will change the side of the platform it’s on in respect to the actual player sprite moving. I did this by setting the position of the rectangle to become below the platform and above the platform based upon the playerState.

Callum Rosewarne-Reece (ROSCS004)

# Game Score

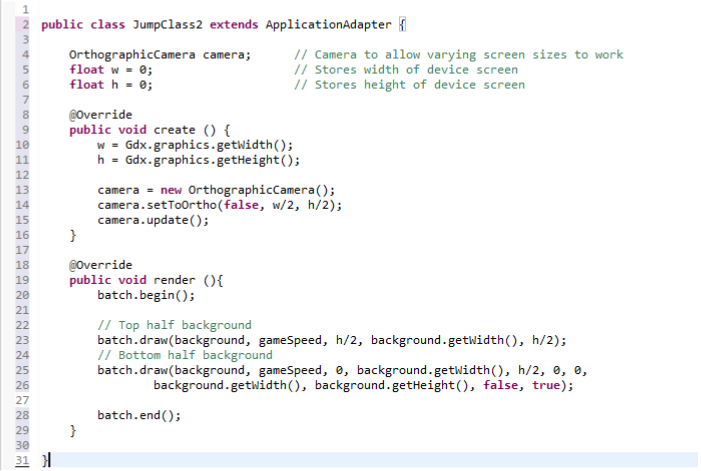
The code for making the game score appear on the screen was written by myself. There is a variable in our code that increments every time the *render()* method runs. This ends up increasing to roughly 60 for every second (that’s around how many times render should be running per second). I deemed this to be too fast for the score, so I divided that number by 4, so that the score won’t go up so fast when the game is played. You can see the code for the score below (with other, unrelated code taken out of this screenshot just to make it easier to read).



One of our major user goals we envisioned from the start of working on this assignment was that since this is an endless/infinite runner game, the key goal of the same is to try and stay alive for as long as possible. Adding a score feature to the game allows players to try and beat their scores from their previous runs in the game, and gives the users a sense of achievement and accomplishment if they manage to beat their own times on each playthrough. If this were to become a proper game on the play store, the game could integrate with the Google Games services and allow friends and people across the world to compare their scores from the game.

# Camera

I wrote the code for the game’s camera. A camera is important in mobile games, especially for Android games, because there are a multitude of different Android devices that exist with varying screen sizes and resolutions. Running a game without a camera on a large device compared to a small one, will likely mean the game will be too small to fill up the screen. A camera is used to make sure that for whatever device the game is run on, it will fill the dimensions of the screen properly.

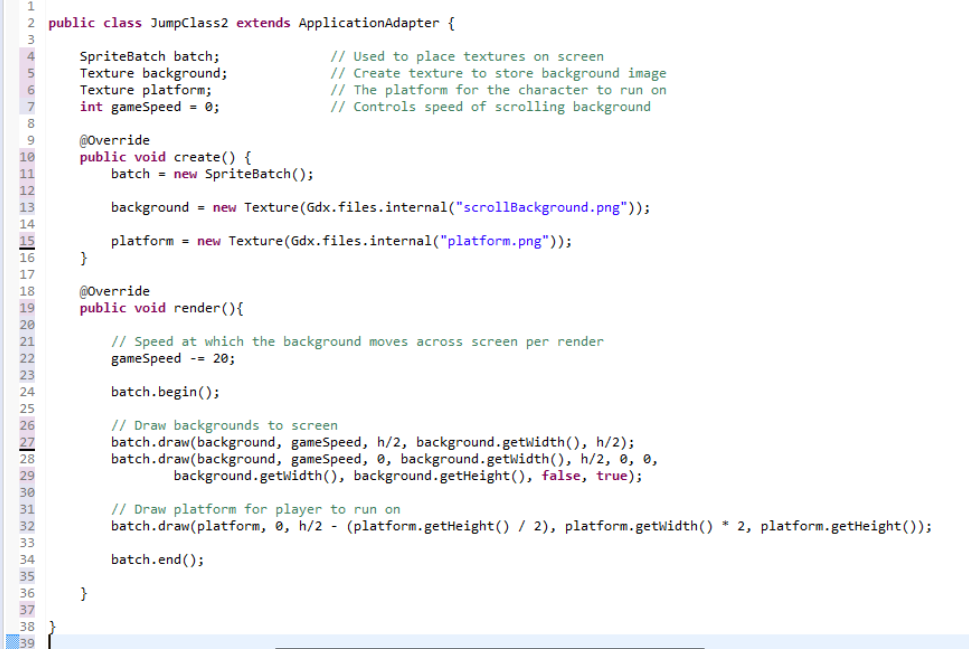


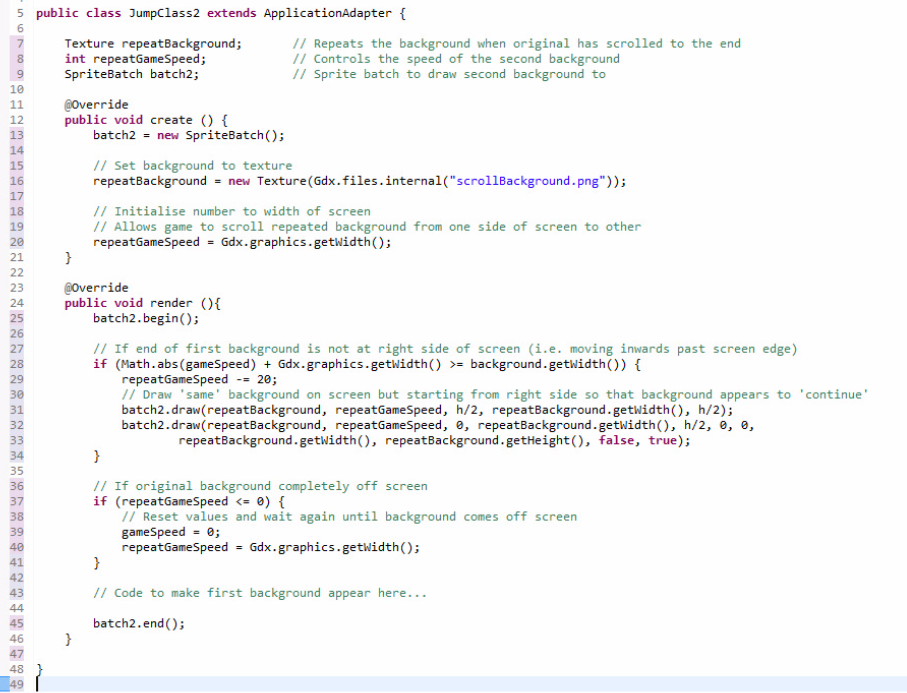
# Adding & Repeating Backgrounds Vertically

I talked before about how I wrote code to make the background on our screen move in the left direction to give the illusion of player movement in the game. Our game contains a flipping mechanic, where our player can move from the top half of the screen to the bottom half of the screen to avoid getting hit by deadly blocks. The game is obviously playable when the character flips to be ‘upside down’, so a background was needed there too to make the game functional. This involved flipping the background, but still having all the same movement properties, like moving it to the left of screen.

An added challenge came with this, in that the backgrounds were rightly so moving to the left, but once they reached the end of their width, an empty game screen would appear behind where the background previously was. This meant adding a copy of the original background, and temporarily moving it across the screen until the first background had made its way off the screen. The temporary background is then removed from vision, and the original background (once off screen) has its x-coordinate set back to 0, meaning it is free to move to the left again. The way this was written means the user doesn’t even notice any of this – the background appears to repeat indefinitely until the player dies.

I also created the platform that the player runs on in the game. It’s only simple in this version – a simple rectangle shape representing grass and dirt. The code for the backgrounds and platforms are shown below.





# Game Speed

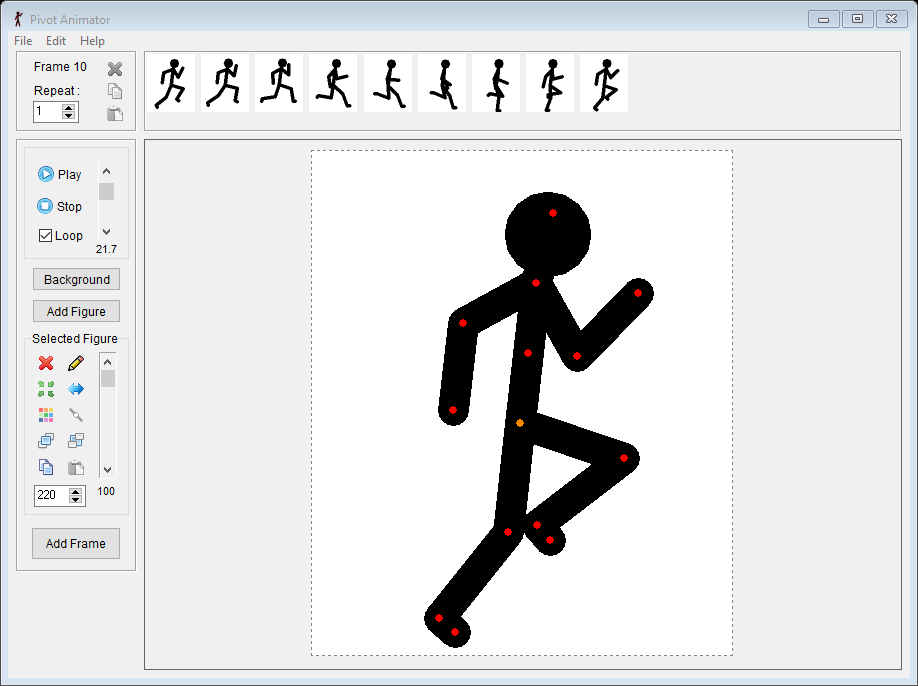
The game speed is determined in our game by a variable. The need for this began when we required the background in our game to scroll across the screen, to give the sense of the player character running in the game (since the character stays fixed on the screen). The game speed in relation to the background is used to move the background towards the left of screen every time the render method is called. The base rate we have set for this game is -20. This means that every render, the background on the screen will move 20 pixels to the left of screen. Deducting -20 each render means we can set the background’s x coordinate to be that number each time the game renders, meaning it will move across the screen for the player. The code for using the game speed is

Zhang Fan (fanzy009)

My tasks were making all sprites and import them into game, lastly switch those animations between different player state.

# Sprites

Firstly, we discussed using sprites from online resources. But we couldn’t find suitable sprites, because we required run, jump and flip animations. Therefore, we decided making sprites by ourselves.

I started using Adobe Illustrator to draw stick man. Soon, I found it’s impossible for me to make an animation in this way. I am not an artist. After Google, I found a software named Pivot Animator (<http://pivotanimator.net/>). It is easy to make stick man animation using this. 

I made some animations through this software:

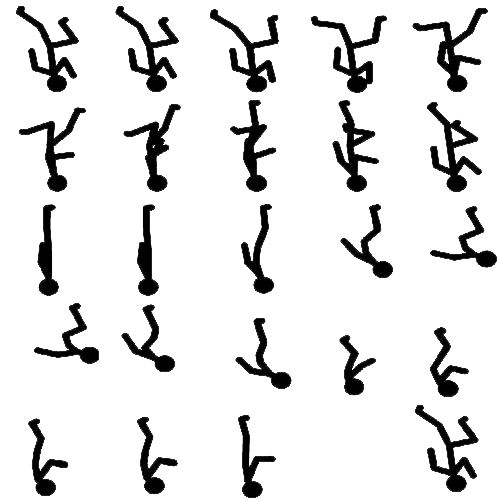
1. Run
2. Jump
3. Flip
4. Jump when flip over other side
5. Run when flip over other side

I exported those animation as png files. And group them into a sprite sheet using Adobe Illustrator.



Run animation

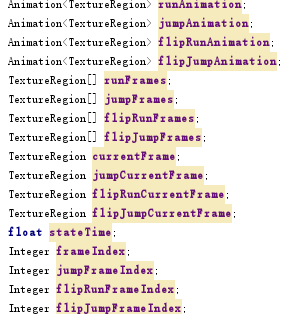
Jump animation



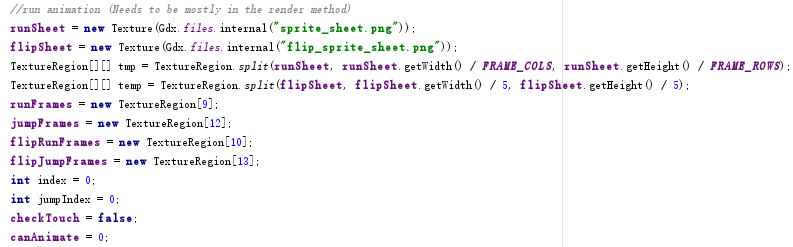
Flip\_run animation

Flip\_jump animation

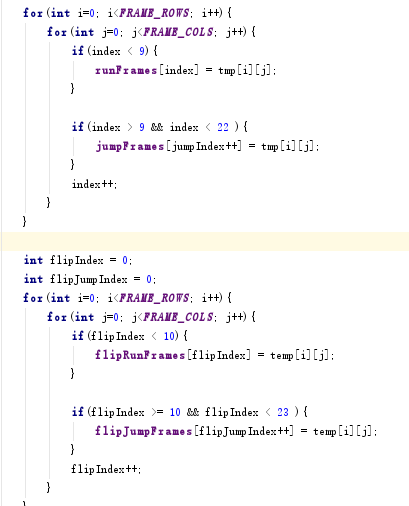
# Import into game



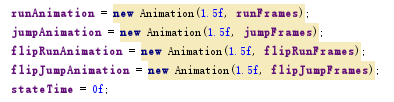
I followed practical instructions and import into game.



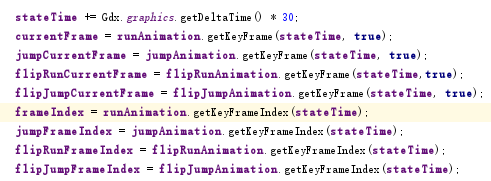
Split the whole sprite sheet into a set of texture.



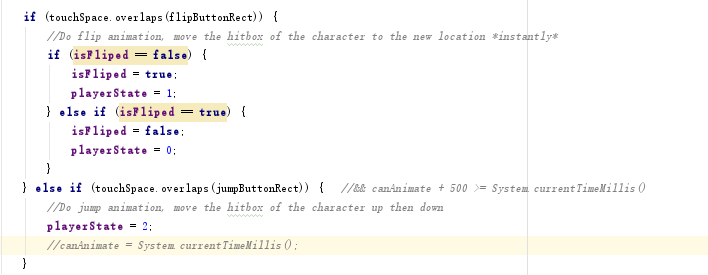
Use for loop to extract frames from the set of texture.



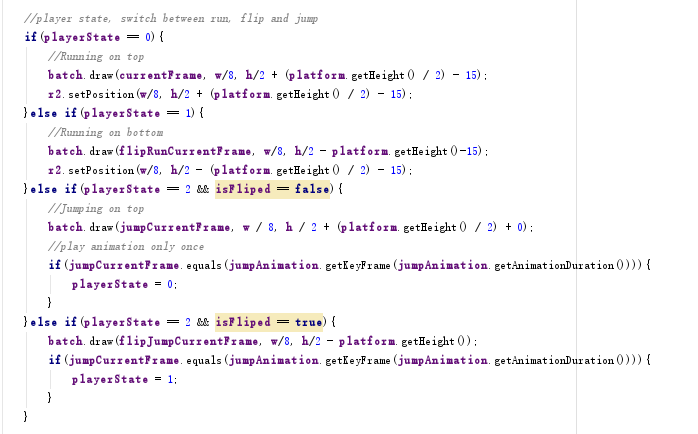
Construct a new animation and initialize the game time variable, 1.5f this float represents the time each frame of the animation will stay on the screen.



Using animation methods to render frame on the screen.



checkTouch is Austin’s work, when screen is touched, now check the touch location overlaps our buttons. Here is my code, I set player state to different int numbers. If the character is at top, we set isFliped equal to false. If the character is at bottom, we set isFlipped equal to true.



Here is the most important part of transferring animations.

Number 0 represent animation, 0 means run animation, 1 means flip animation, 2 means jump animation.

When player state equals to 2, we also need determine whether flipped. The next if statement is to stop animation when last frame of the animation is played.

# Meetings

#### April 7, 2017

Attendance:

Austin, Callum, Toby

In this meeting we decided our roles in the assignment. We spoke about how we would implement it and decided what we would have finished by the end of our first sprint.

### Minutes

* Toby will be responsible for sprites and getting them working in the game
* Callum will be responsible for the background and platform
* Austin will be responsible for collision detection, touch detection and getting the blocks drawn on the screen
* We will meet again in 2 weeks and give new roles based upon the amount of work done

#### April 21, 2017

Attendance:

Austin, Callum, Toby

In this meeting, we worked out how far we were in the assignment and gave each other new roles based upon our work in the first assignment. We also ensured that we would meet again within a month to discuss further progression.

### Minutes

* Toby finished creating the sprites, he will now put them in the game
* Callum found good backgrounds and now has to put them in the game and make them rotate
* Austin has place the blocks in the game and will begin collision detection
* We will meet again in a month for our final meeting, and finish the assignment

#### May 21, 2017

Attendance:

Austin, Callum, Toby

In this meeting we discussed how we would finish the assignment on time and what we had left. We handed out roles and decided we would meet again to complete the document and PowerPoint on the 6th.

### Minutes

* Toby has put the sprites into the game and we can now do collision detection with these sprites
* Callum has made the background repeat and it looks nice
* Austin has finished collision detection and will now begin touch detection and tweaking the collision detection so it works with a smaller hitbox

#### June 6, 2017

Attendance:

Austin, Callum

In this meeting we finished the assignment and finished the document. We also worked on the PowerPoint and discussed how the assignment went

### Minutes

* Tomorrow we’ll submit the assignment
* We’ll meet before the lecture on Thursday and practice our presentation