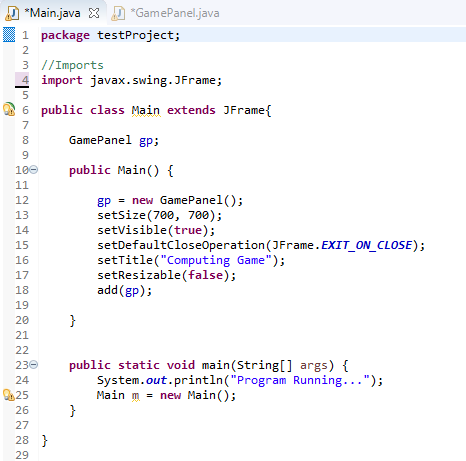
**Render Engine Documentation** – Jake Cooper

To start this off, I basically copied line for line a tutorial from Youtube ([TheJavaHub](https://www.youtube.com/channel/UCBgSCTqTUk5PrItkQF4bjSA)) that I’ve used before to write this Rendering Engine. Its nothing fancy like you’d see in a professional game, but it does the job perfectly for the amount of effort it took to write. It is however quite complicated, so I will attempt to break it down in this document.

­

This first file is Main.java. When the program starts, this class is ran by default so is where everything begins.

The first line is a standard is java that just defines what package the file is in. The imports are exactly the same as imports in Python and allow me to use different Libaries (In this case I’m only JFrame which is used to create a window



­

GamePanel refers to another class which I use to do the majority of drawing onto the screen. I could draw directly onto the JFrame but GamePanel offers a bit more flexibility, so I place the GamePanel onto the JFrame instead [ *add(gp)* ]*.*The rest is fairly self-explanatory other than [ *setDefaultCloseOperation()* ] which simply makes sure the program stops running after the window is closed.

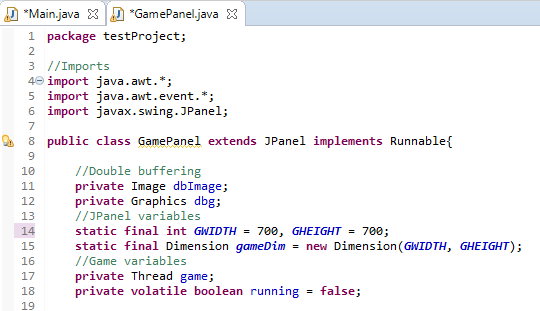


This method is what Java first reads when it opens this file as it’s a standard method. Since I want my program to run the contents of Main(), I simply create a new instance of it in this method so that it will run [ *Main m = new Main();* ]. I never have to actually call “m” but a placeholder variable is needed for this to work.



This next class is where the majority of the Render Engine sits so its where it starts to get quite complicated and is quite long. As I said earlier, I’ll attempt to explain it best I can.

This file is GamePanel.java. It gets called right at the start of the program inside Main.java of which it will start by running [ *public GamePanel()* ] so I guess I’ll start from there but before that I’ll go through the variables and starting of this class.



[ *extends JPanel* ] means that I can use the methods within JPanel without having to define it before in something like [ *JPanel jp = new JPanel()* ] which saves a lot of time when typing this all out. JPanel is a library that can create graphical windows that can then be placed onto a JFrame as said earlier

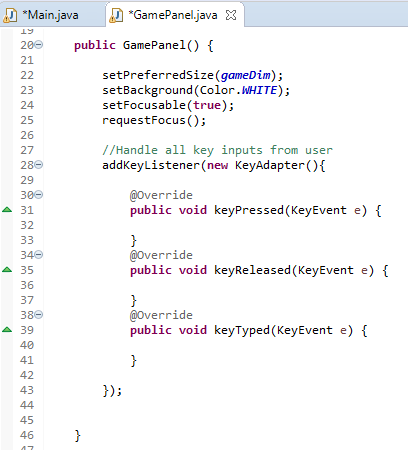


[ *implements Runnable* ] is a bit harder to explain but in short it allows me to use Threads and the method [ *public void run()* ] which is basically a built in while loop and will repeat for the entirety of the programs up time.

Double buffering variables are used later in the class to create a buffered frame and display the previous one. Using Double buffering hopefully means there will be less graphic tear and glitches when running the game as the next frame is rendered before it needs to be drawn.

JPanel variables are simply used to define the width and height of the Panel (This is the same as the size of the window created by JFrame) and then turns this into a dimension as JPanel only accepts them as a Dimension. In case you don’t know, the term [ *static* ] means that the variable will stay the same through out all instances of JPanel being called. JPanel is only ever called once so this isn’t entirely necessary but with larger projects, calling these variables as static will save quite a bit of memory as each instance of the class can use the same physical location of the variable in RAM. [ *final* ] means that the variable cannot be changed or modified, again saving memory in larger projects (and has the added benefit of being a fail safe if I accidently change the variable somewhere else in the program)

Game variables are used to keep track of and initialise the game state. The [ *Thread* ] is created here so it can be ran further down in the program and the Boolean [ *running* ] just helps keep track of weather the program is running or not in case some unforeseen error occurs.

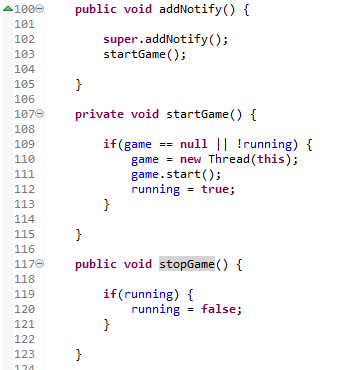
This is the first method to be ran when the class is called over in Main.java

These are methods inside the JPanel library but since we said our class extends from it we can use them as if they were in our own class. All of these methods explain themselves really.



[ *addKeyListener()* ] is a method within a library in [ *java.awt.event.KeyListener* ] but since we imported all of java.awt.event at the start then this is included in that. When a key is pressed, released or typed(tapped) then the keyListener will call the following methods with the specific key that was changed as the parameters. This occurs as long as there is an instance of this class so practically this means as long as the program is running.





The next method to be ran is [ *addNotify()* ]. I don’t fully understand the concept behind this but when the JPanel gets placed onto the JFrame, this method is called to Notify the JPanel that it is indeed on the JFrame. So when all of that has been confirmed, we start the game by calling [ *startGame()* ]



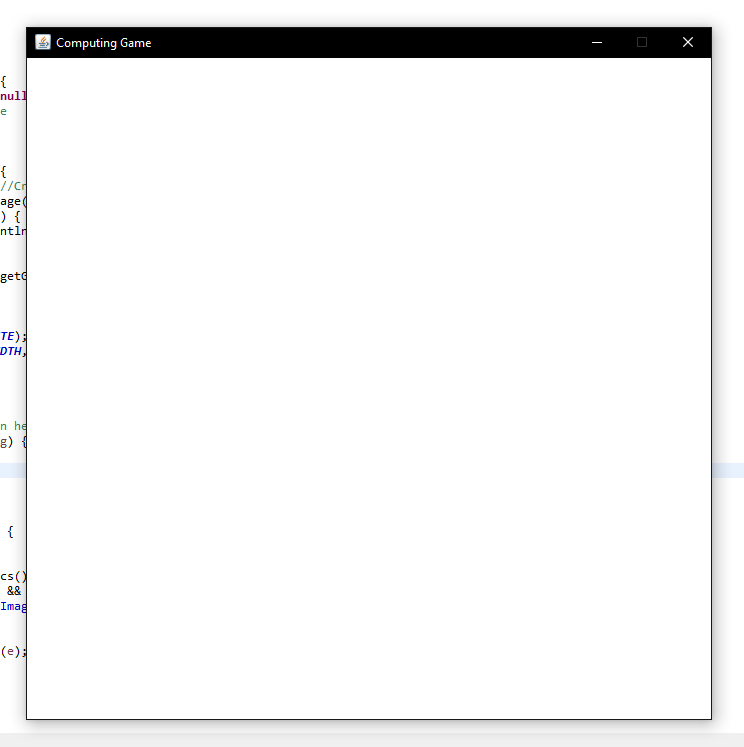
When this method is called, it first checks whether the game isn’t already running for some reason by making sure the Thread doesn’t exist and the running variable isn’t true. Once they are checked, the Thread can be created for this class and started (this will direct us to the [ *run()* ] method). [ *running* ] is then set to true to make sure everything is kept track of.



This public method simply sets running to false when called. Its public so can be called by any aspect of the game such as a menu. The only real loop that uses this variable is in [ *run()* ] which is shut down the rendering of the window before closing it.

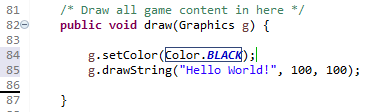
General Workflow loop of the program

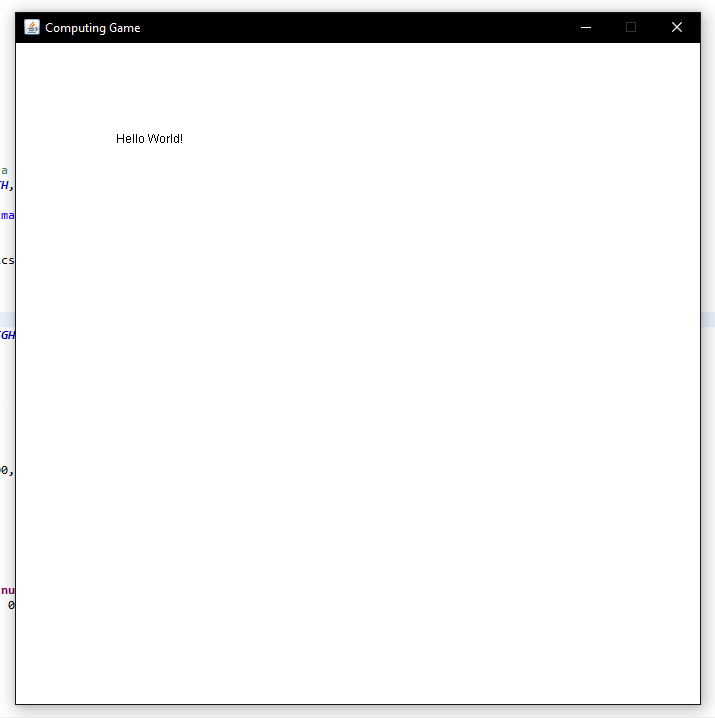






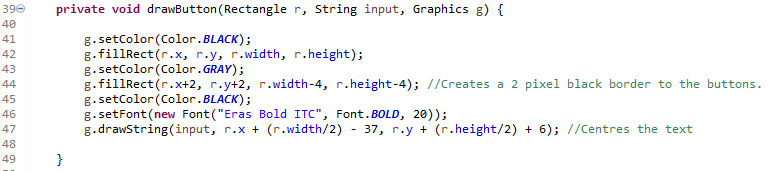
Showing off the results of the engine



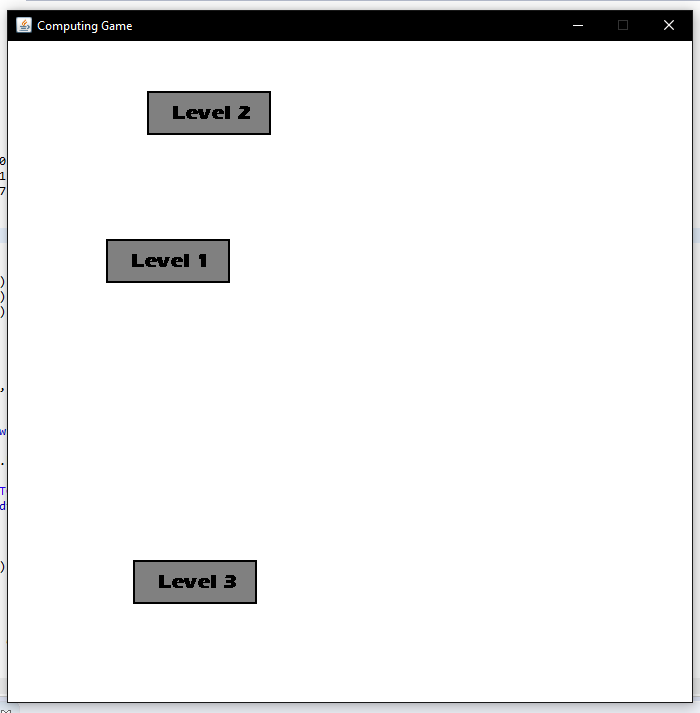




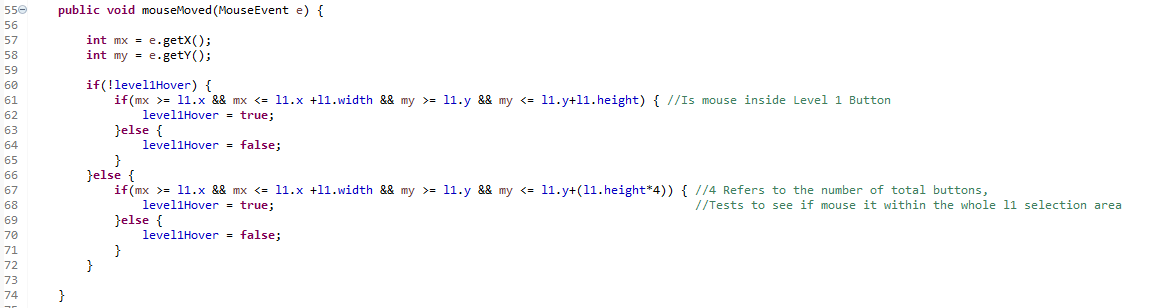
Drawing the buttons







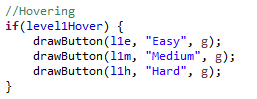
Hard Coded Hovering method



Have to copy past this for every button but that shouldn’t be too hard considering I only have to change the numbers.

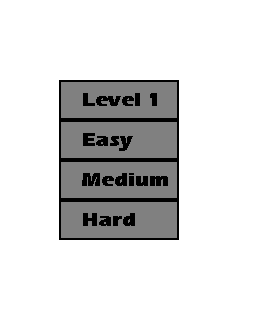
When dropdown isn’t already out, it only checks for when the mouse is over the button, when the dropdown is out it checks to see if the mouse is over all the buttons, so they don’t disappear when you try to click them.

Actually drawing the dropdown



Result, gif - <https://gyazo.com/2501e637ef536c7afc2f06d7419f76be>

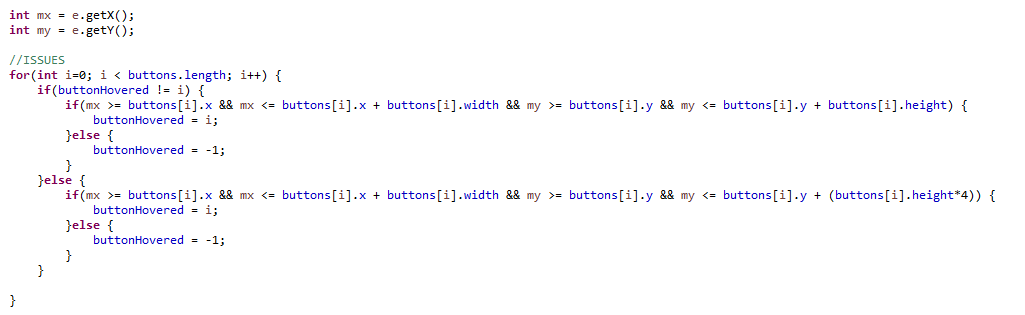
Screenshot of what the dropdown looks like



**Detecting mouse hovering and dropdowns**

Method 1

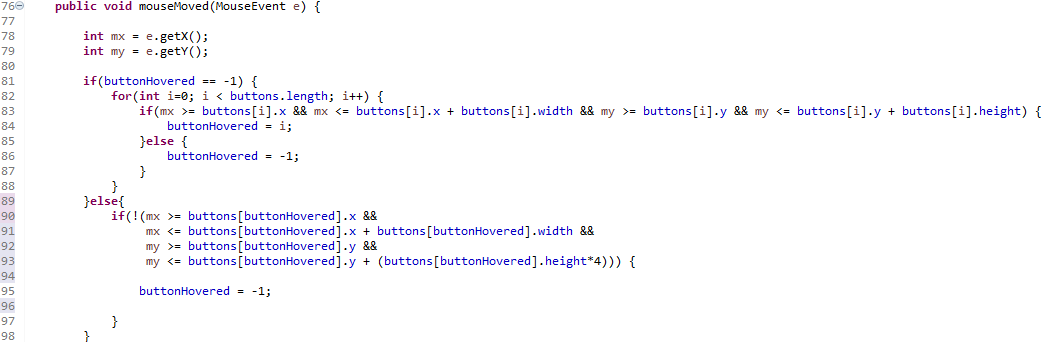
Decided to try cut down the code to make it expandable with all buttons.



This only worked with one button and would not account for when the dropdown activates.

Result from Method 1 - <https://gyazo.com/8aa793aa0ff5b6de372bcb8cb8e11e5c>

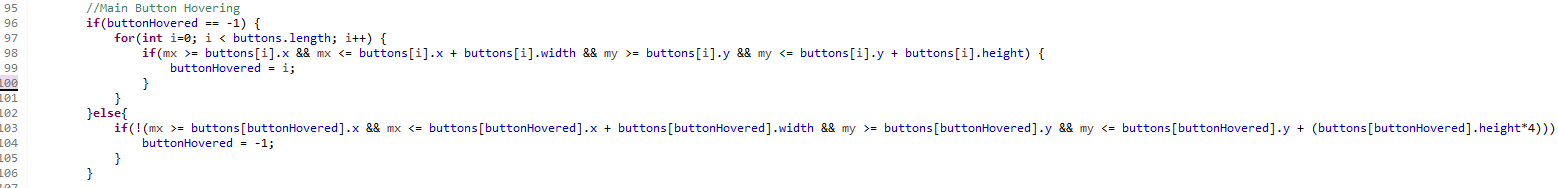
Method 2



As shown in the results, the dropdown now works as expected but still only for one button

Results from Method 2 - <https://gyazo.com/82fdbc3c7d216a478f16b746a9749003>

Fixed Method 2

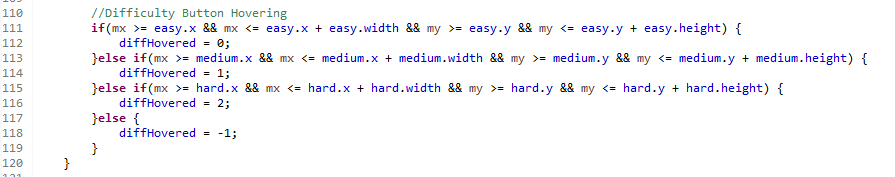


Essentially what I did wrong is that the detection of hovering over a button at the start would be cancelled out by later buttons in the array because of the else statement.

Result from Method 2 - <https://gyazo.com/be9407ecc1d134252540881237b3bd6c>

Detecting mouse hovering for difficulty buttons

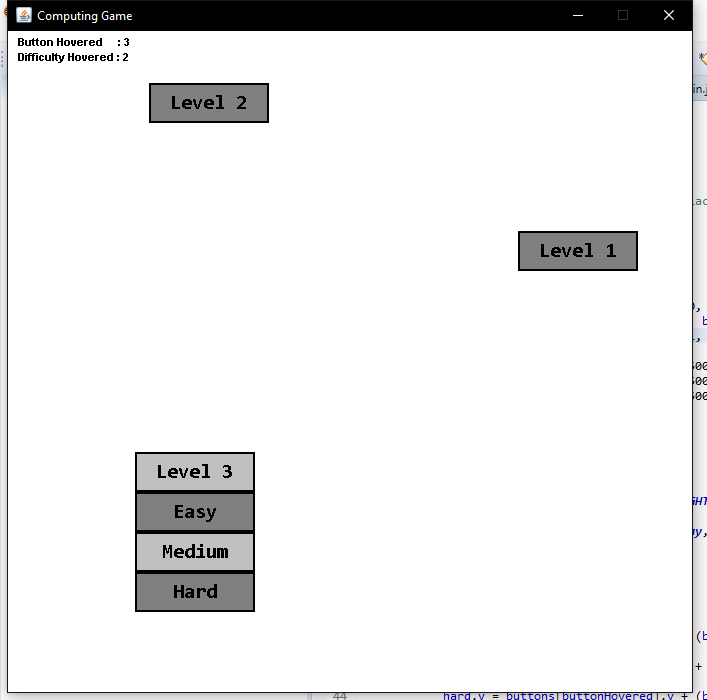
Detected here



Drawn here



Result + gif - <https://gyazo.com/a261fe9f1a571a36f23972f0976f7c76>



Selecting Levels/Difficulties

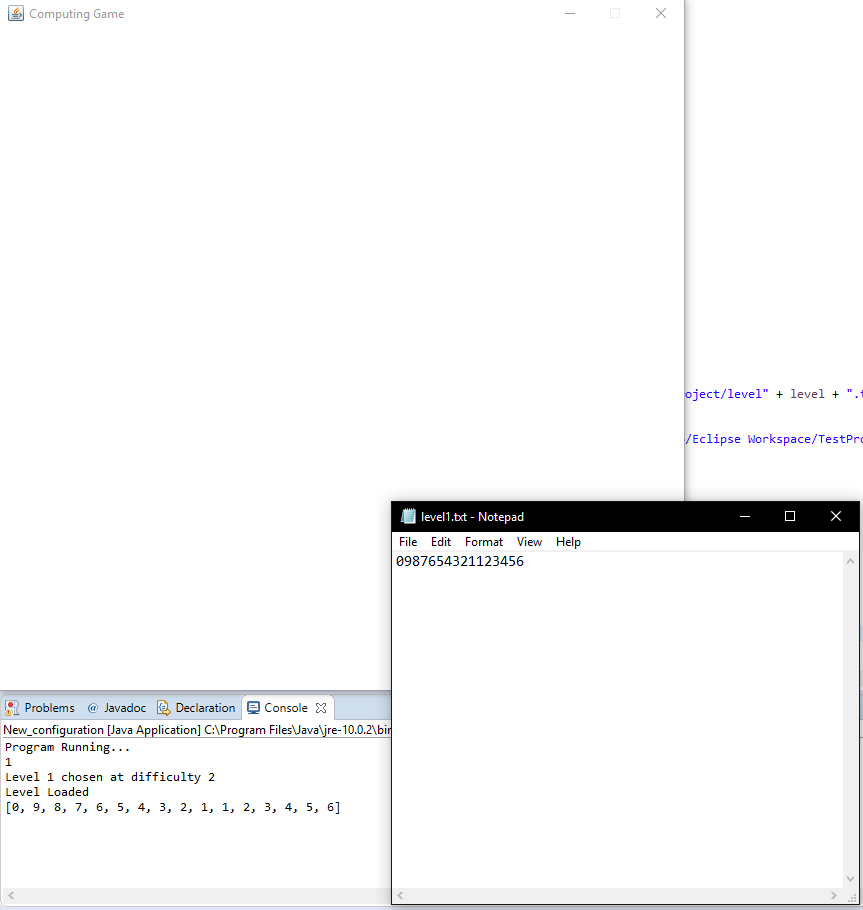


This changes the level and difficulty when buttons are clicked and then loads the appropriate level and diff onto the stage which will then get drawn. (The system printing and ( || true ) are just for debugging)

Resulting gif - <https://gyazo.com/8a2a98417c4a143436bddcd97fafd60b>

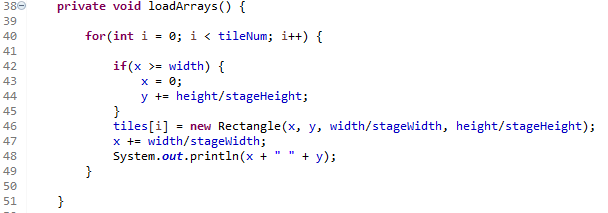
Loading level file into a String Array

So for ease of changing etc. I want my level file to be in a separate file. Its format will just be a bunch of numbers where there position in the file is there position on the screen and their value is the type of tile they are. Therefore I had to write a small method in order to turn this file into an int array that I can then draw onto the screen.

Result

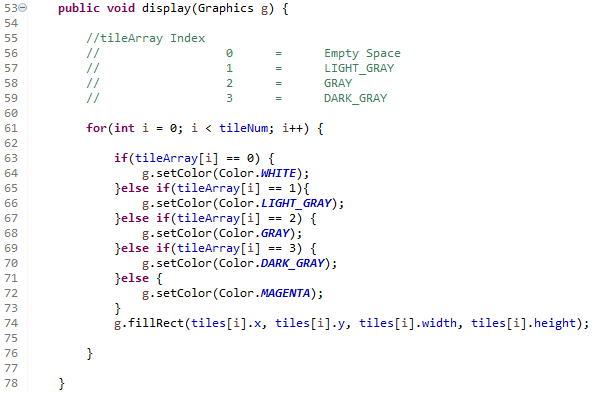
Drawing the Level

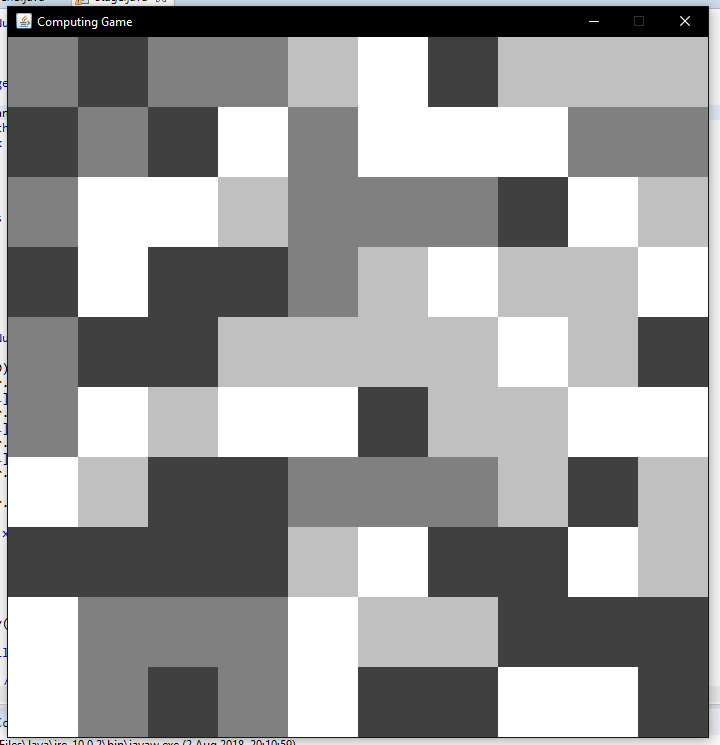
There are a few parts to drawing the level, first is deciphering where all of the tiles go (which I’ve made scalable for bigger windows or denser levels) which is done in this method



This just goes though a Rectangle array, setting each one to a position on the screen relative to the size of the screen and number of tiles intended to have.

Next I need to draw the Rectangles on the screen. For the time being, I have them draw as different colour for different tiles but when I get around to skinning the game these different colours will correspond to different tiles (floor, wall, etc.)



As you can see, the colour is changed depending on what value is written in the file. The result looks like so

As a side note, I’ve created a small excel sheet to create levels easily.

Also just for testing purposes, I’ve used random numbers to decide what each tile is. This will obviously be in the layout of an actual level when I get round to designing them.

Here are gifs of me opening all 3 Level Files (all just random numbers)

Level 1 – https://gyazo.com/243d3354f0f28bc8705400b5fd61e51e

Level 2 – https://gyazo.com/2891724d28004f278b8d630a7a37a1b0

Level 3 – <https://gyazo.com/3a30ec7f17bbd5e0509ba369acf99225>

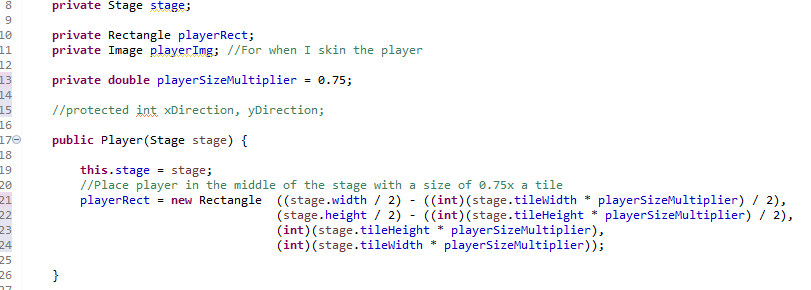
Making sure the game runs at a constant rate

So if I were not to make this change, the game would update and render faster on better computers meaning gameplay would change from computer to computer. Doing this change stops this by setting a fixed rate the game runs at and sleeping the thread if and when it goes to fast or slow.



No real results to this but when playing on different computers it will make a huge difference

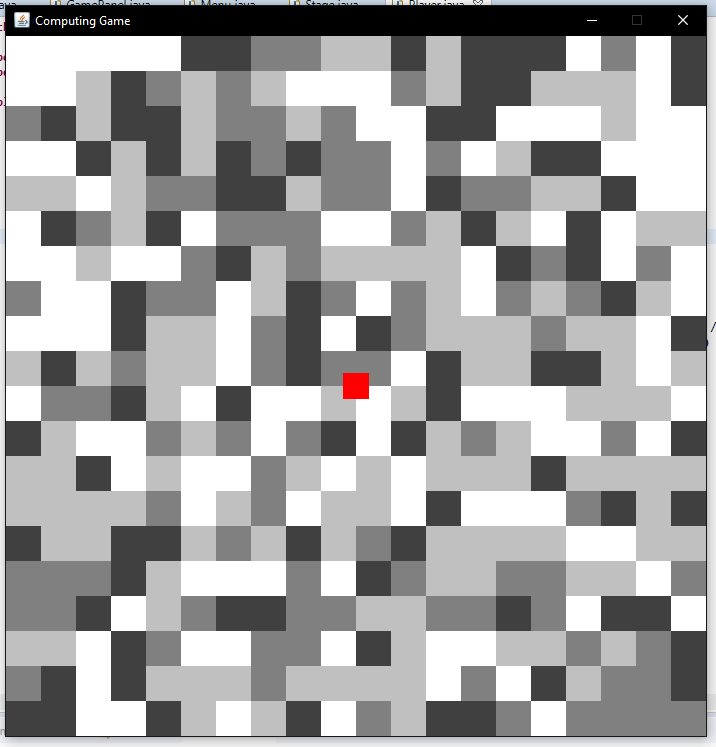
Creating Player in the middle of the screen



This is inside Player.class and creates a rectangle that is ¾ of a tile big and is placed in the exact middle of the screen. This bit of code does exactly that. Once again I am drawing the player as just a red rectangle for now but skinning over this rectangle is not hard at all so I will do that once I finish the mechanics

The reason I put him in the centre is because I want all games to start from the centre meaning that Ill have to make sure when designing levels to keep these 4 squares open.

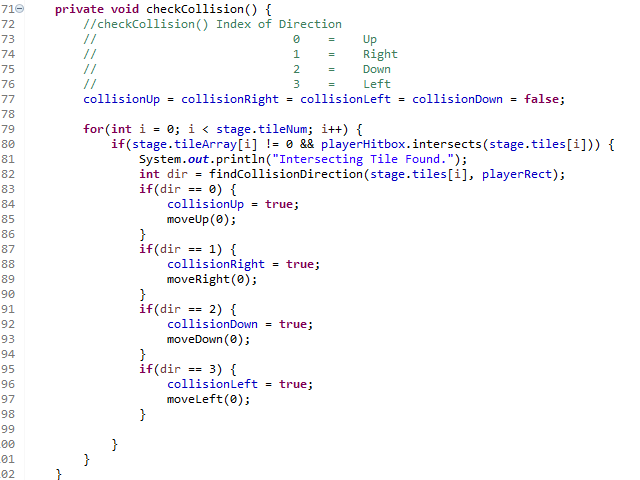
Results from this



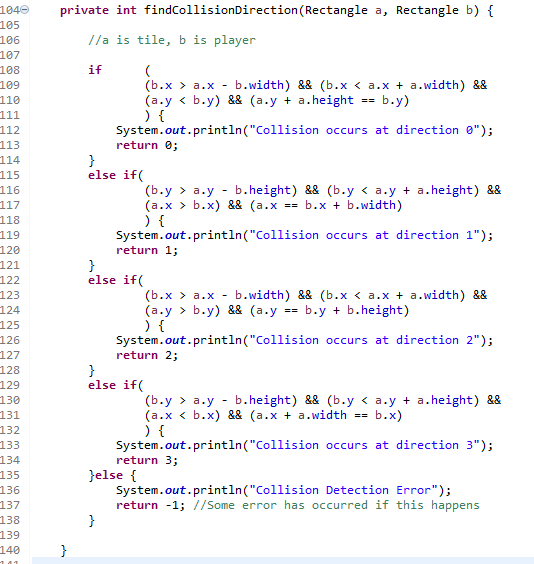
Make shift collision detection

Simple collision detection is fairly hard to find online so I decided to just make up my own. This however means that it is a bit sloppy at the moment (if you try hard enough you can clip through the corner of tiles) but for the moment it does the job as intended.

To do this, every game cycle I check for any collision on the player and set some Booleans to show what direction that collision is in



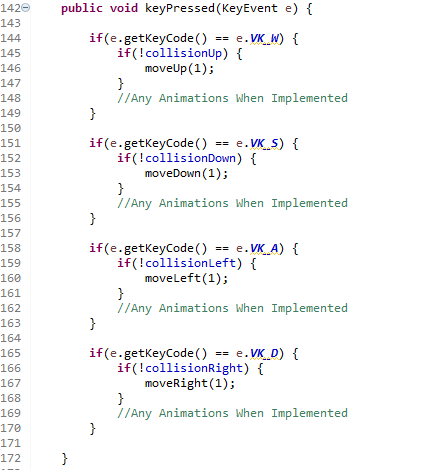
This sets all collision Booleans initially to 0 and then iterates through every block on the map, every game tick, to see if the players hitbox is colliding with anything. If it is, that specific tile and the player are passed into another method to find the direction of which that collision occurs at

This is the main bulk of detection. The upper part of the if statements are making sure that the player is actually within the bounds of the tile. I was having trouble where the player was getting stuck along edges but this fixed it. The second line uses a couple of inequalities to decipher which side of the player the tile in question is.

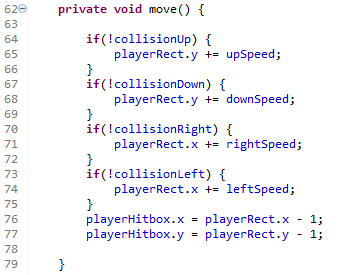
The results of this can be seen in this gif

<https://gyazo.com/c02f5a0c6084b5931ae8f0bb0d4e0f63>

In my move method, I just decide whether the player is allowed to move or not depending on if there is a collision in that direction. This is shown in this little bit of code on the next page.



This is an extension to this above bit. There were issues with the movement from above as even the player hit a wall but then moved to a position where it no longer blocked them, they’d have to repress the button to go in that direction. To combat this, I’ve removed all mention of the collision Booleans and only used them in the move() method as shown below



This means that once the block is no longer in the way, the player will continue to move as you would expect. There is still the issue to moving into the corner of blocks but its quite hard to do and with proper level design it should become a non-issue

Result gif - <https://gyazo.com/1a07d4cc549aa5f45aee5fb81bbbd6f3>

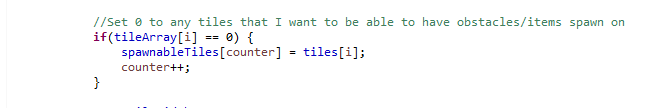
Preparing to spawn obstacles and collectables

So to be able to spawn obstacles and items to collect at certain intervals, I first have to keep track of every frame since the stage is loaded. This is really easy to do since every frame the display() method is called so at the end I can just put



Now I am prepared to spawn things on certain intervals

To spawn these items, I also need to be able to find out what blocks can actually have items spawned onto. To do this, in my loadArrays method I have added another statement that puts blocks into a separate spawnableTiles array if they are of a certain tile type. This can be seen here

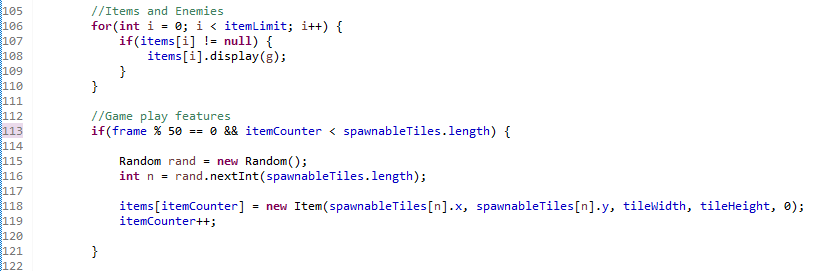


Thanks to these two parts, I can now create a enemy and item class and call them every lets say 1000 frames and have them only spawn on spawn able blocks.

Item Class

This is a simple class to start off with as all I want it to do is be able to be summoned, a bit further down I can add the logic for it being picked up etc.

I currently have it set up so that the item can be summoned from one of two colors with magenta being the (something messed up on summon) case.

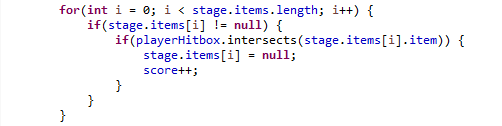


This basically means that every 50 frames, an item will spawn in a random white space.

Gif of this in action - <https://gyazo.com/2ae1954f33cc9ebd6676f4d63f1802ac> - Obviously this will be a lot slower in actual gameplay

Collection of the items and adding them to the score

This was a simple addition and just allows me to pick up the items and have 1 score added to my total (temporally at the top left with debugging stuff) as well as remove the item afterwards to save on a bit of memory.

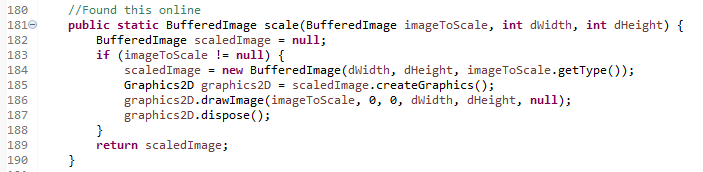


This is simply all the detection that had to be done. Heres a gif of it working - https://gyazo.com/dbdd4c2a0a180464ce353d60bd52bf34

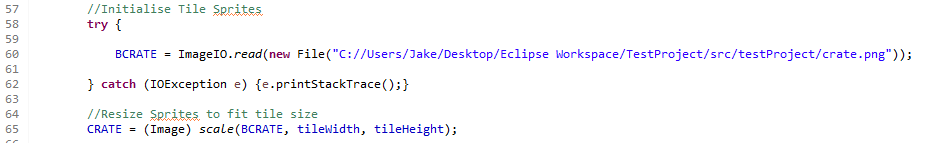
Resizing Sprites

I didn’t make this method as I found it online but essentially what it does is takes an Image and dimensions as an input and spits out the correctly sized image as an output. When you draw an image java will only draw it the actual size of the image so resizing it is necessary.

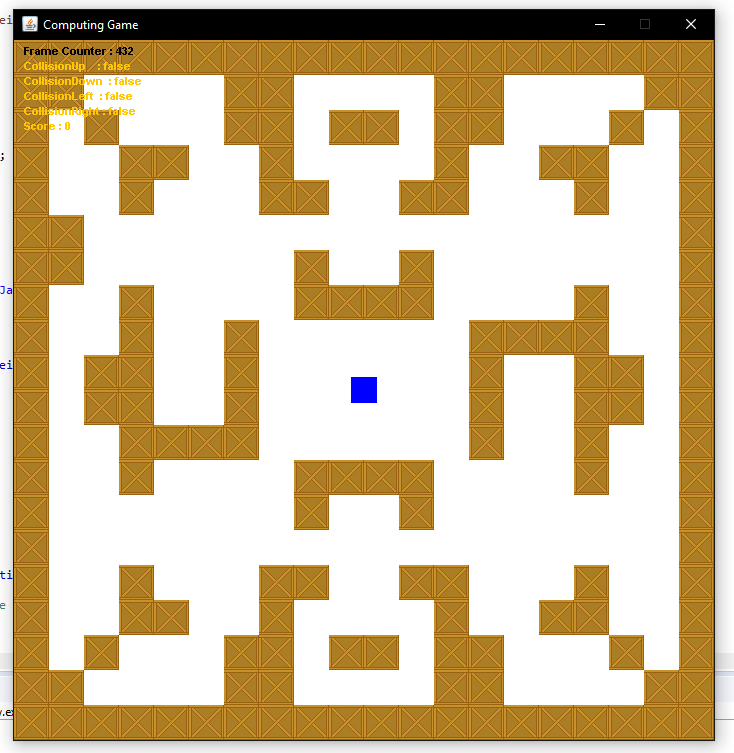
This is what the method looks like although I have little knowledge of the technicalities behind it



And here is the application of it



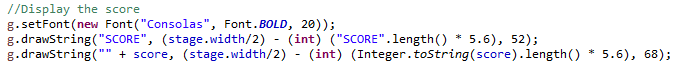
I can then just draw these images over the top of my tiles. Here’s what it looks like using mock up crates I quickly made. The original resolution is 700x700 but they have been downscaled to fit the size of the playing area.



Displaying the score

Small addition but had to add a little algorithm in order to keep the score in the centre of the screen. As a side note to this project, I’m noticing more and more parts of my game aren’t scaling well with a change in game size. This isn’t really the biggest problem as I’ve decided to keep the game size at 700x700 but it means a bit of the work I put in earlier on is less useful because the rest doesn’t scale. A couple examples are the level layout (if the size of the grid doesn’t scale exactly to the size of the window then things freak out) and the player movement which is a constant 1 pixel per second so they move faster on smaller windows and slower on bigger windows.

Anyway, here is the score displaying lines

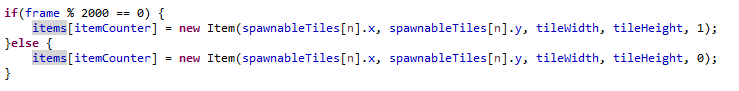


To show its centred always, I’ll screenshot it at 1 digit length and 2 digit length

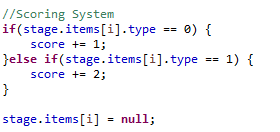


Adding a different type of item with a different score to show of the mechanic

So the way I’ve set up the items, you can spawn different types of items. To show this off I just have it spawning a special item every 4 item drops. This is shown here



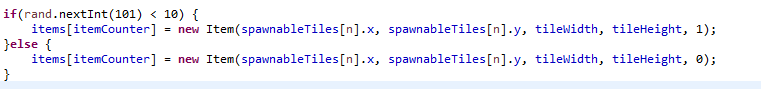
I can also add that these special items reward more points. This was also fairly simple and can be seen here



This means that normal items will reward 1 point and special items will reward 2 points. This is also easily expandable to more types of items.

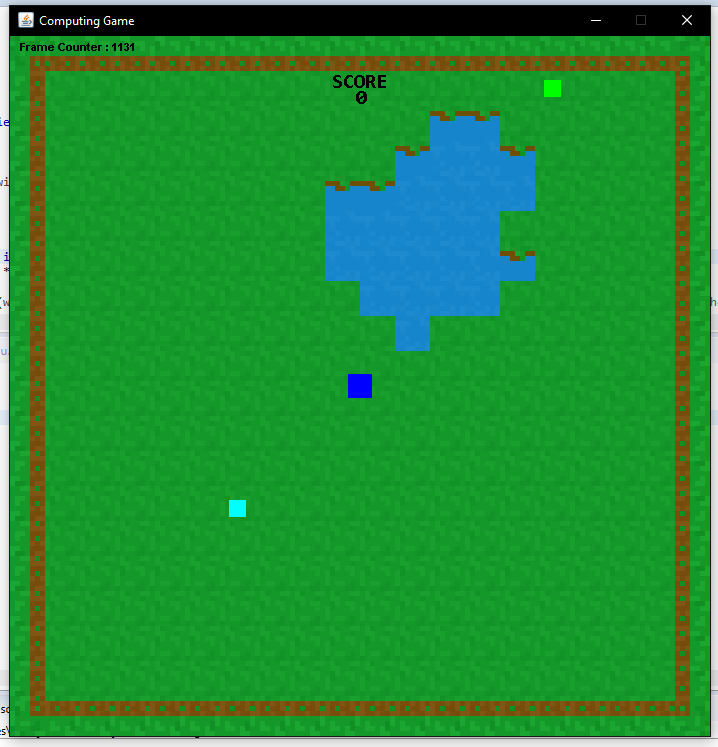
Gif to show off the special tiles rewarding more points - https://gyazo.com/e6ba22fc6a440a0ee47f600e289db859

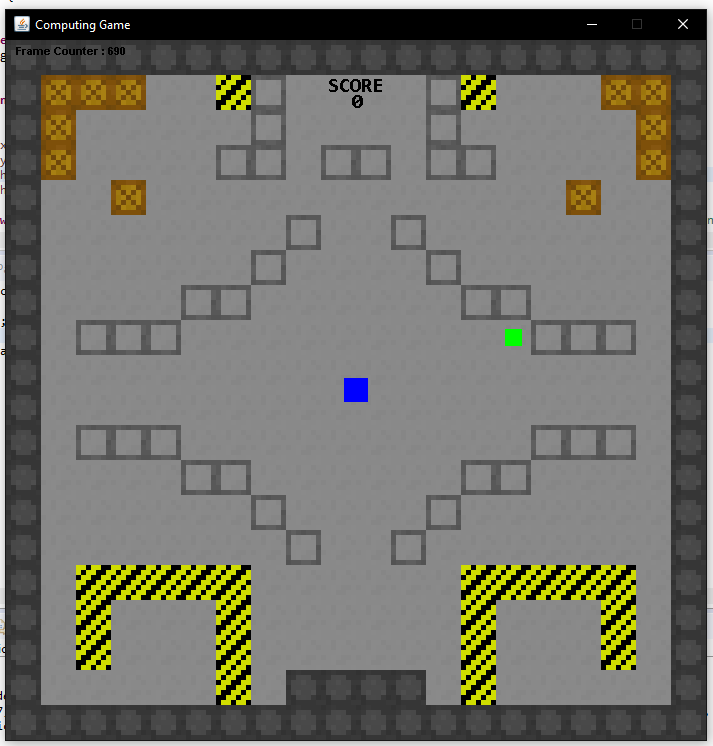
I can also do it as such to make the special tile spawn a certain percent of the time. In this example one will spawn 10% of the time.



Progress on Level Design and putting textures onto the game

I decided to take a break from coding since it burned me out and dedicate some time to creating textures to fit the game. I’ve decided that each of my 3 levels will be based in different areas. So far I’ve decided on 2, one being an industrial theme and one being an outdoors theme. Using these two ideas, I’ve created a bunch of tile sprites and put together 2 (very first design) stages to rate the feel of it.

This is the current stage 2 And there is the current stage 3



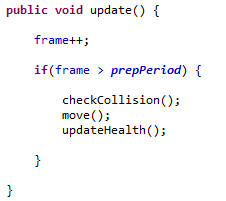
These are by no means finished and especially stage 3 I want to add trees and such to manoeuvre around (as quite clearly its very open). Ill also play around with the actual layout of the stages once I get some kind of obstacle implemented but overall from these two thrown together stages I really like the feel that the game is giving. Its got a colourful and clean style which I think will definitely appeal to the target audience I’m looking at. The pixel art style also gives it simple look which will help keep the game easy to follow.

Creating a health bar

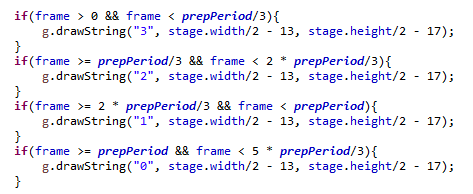
So for my game I’ve decided on two “challenging” mechanics. One will be ‘bombs’ that spawn randomly and explode after a given time, the other will be a slowly decreasing health bar that can be topped up by collecting items. This will hopefully discourage players from just waiting around for items to build up whilst avoiding the bombs.

To do this, I’ve first added a little countdown period when the stage is first opened. This gives the player a little time to prepare and just generally makes the game feel a bit more polished.

First, I updated my update() method in Player.java to both count frames and only start updating the player mechanics after a certain frame period as seen below

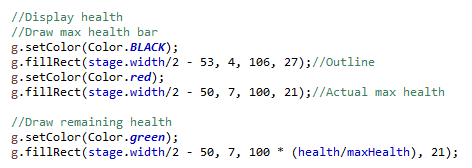


I’ve then set up a few draw calls in display() to show the countdown just above the player. This looks like so:

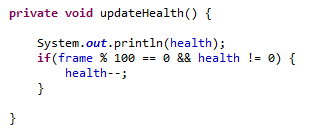


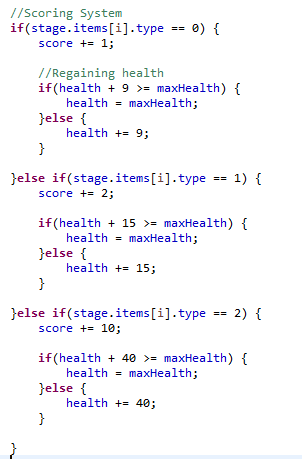
This simply splits the preparation period into 3 chunks then draws a different number on each interval. The player can then begin to move when the timer says 0 of which it disappears after a small amount of time. Here’s a gif of it working: <https://gyazo.com/9187b56b742816492d57d8588dbe5016>

Next I want to draw and update the health so that it slowly depletes but can be refilled by items.

This is for drawing the health bar. This simply makes a big health bar for max health and then a scaling current health bar that overlays on top.

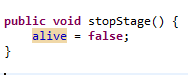
Here’s the code that makes the health slowly go down as the game goes on. The speed that this happens can obviously be changed by changing the frame interval



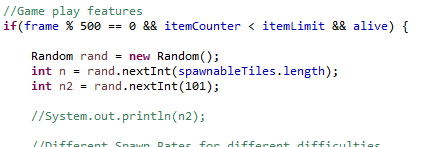
This is for increasing the health once the player picks up an item. When all game mechanics are implemented I can tweak the numbers of this but for now I’ve set them just lower than the loss rate of health so you have to rely on the special types of items in order to actually gain health. This may make the game more luck based but I can worry about that a bit later on.

Finally to finish this off I’ve added this small piece of code. This alive Boolean is public so can be detected by other classes (leading to detecting high scores and settings them etc.) and stops the movement of the player once alive is false. That pretty much concludes the health system I want in the game.

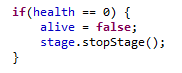
As a little correction, I’ve added a stopStage() method to Stage.java in order to stop the spawning of items once the player has died. This looks like such



Which is used here:



And called here:

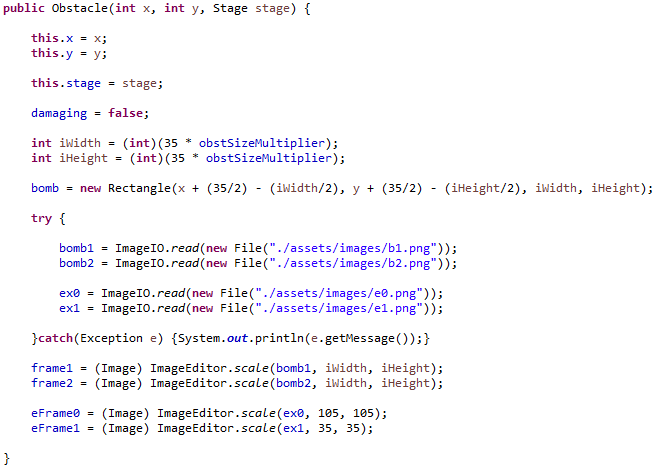


This can be seen here : https://gyazo.com/747d80387a6d530c779a6cc8f0d39bd2

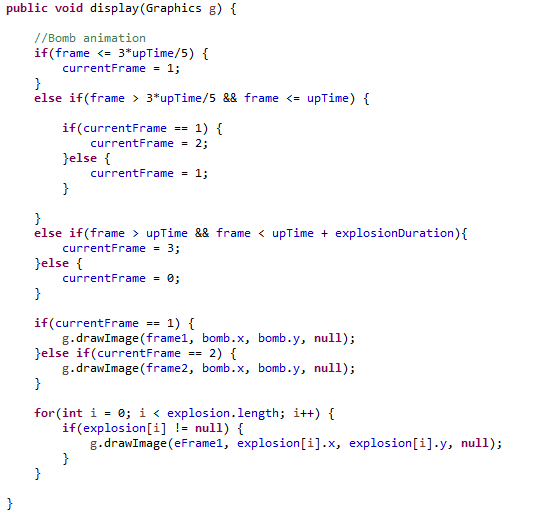
Creating the bombs (Obstacles)

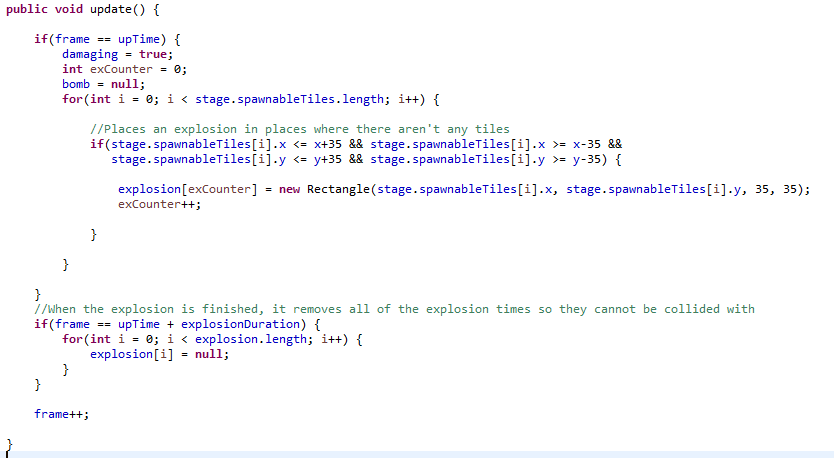
I decided to do this all at once since it was a little hard to keep track of everything so I’ll go through each section I did in sections.

First is the Obstacle class itself. Fundamentally, its fairly similar to the item class just with a few bits added on. To start, the obstacles are obviously created which is done here

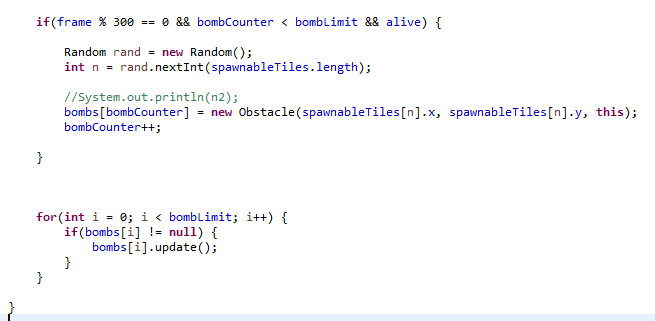
This just places the bomb in the middle of the tile that it is spawned on. One thing to note that is different to previous screenshots is the scale method I’ve separated into its own class that can be called by all the classes. This just means that I can call it wherever instead of having to copy paste it into every file.

Next I suppose would be drawing the bomb itself :

Here I use the currentFrame variable to save lines and changing it changes the image that is displayed. I have it so that in the last 2/5ths of its up time it flashes between two separate images that I have of the bomb (a black and a red version) to indicate that it is about to explode. The bit underneath says that if the bomb is in its time that’s its blown up then show all of the explosion particles which I’ll show how I create next.

So here, once the “bombs fuse” has run out, the class will create up to 9 explosion particles depending on the surrounding blocks around it. Explosion particles because of this wont spawn where there are solid tiles (looks nicer and cleaner). The bottom bit makes sure that all the explosion particles are properly cleaned up from memory once they have disappeared (with the bomb spawn rate the explosions could get in the 1000s if left un removed). It also has the added benefit of making collision with them easy as the rectangles no longer exist.

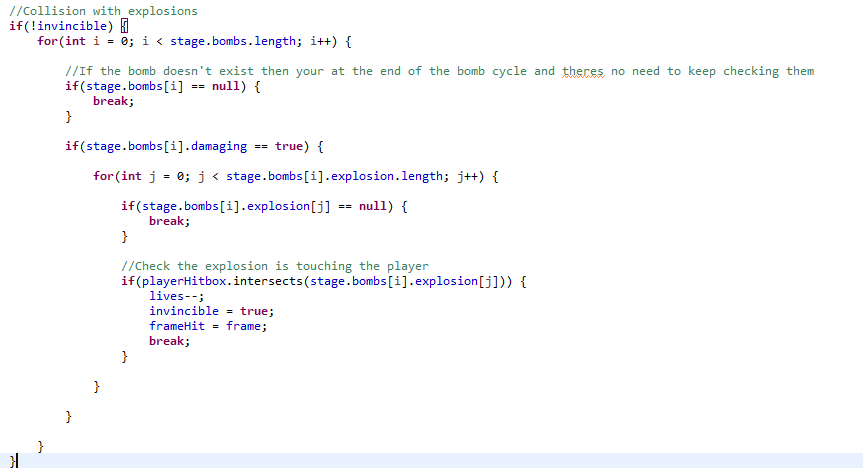
That sums up the Obstacle class, now to where they are spawned.

This is in stage.java and works similar to the spawning of items. Every 300 frames a bomb is spawned at a random spawnable tile. As with items, the bomb is spawned just into “the next” slot in an array of bombs meaning that technically there is a limit to how many bombs can be eventually spawned. For now I’ve set this at 2000 but at some point I may make a method to find the first free space in the array which shouldn’t be too hard. The small for statement at the bottom makes sure that the bombs that do exist are being updated.

A full gif of this in action can be seen here : <https://gyazo.com/1cf29d70f4343be3854f6498c5be5ee7>

Just to show, this is what it looked like without the only spawning explosions over spawnable blocks. As you can see it looks a lot messier and having certain tiles block the explosion makes the game look much better : https://gyazo.com/66953b4d8848c354682b6c643568e8ea

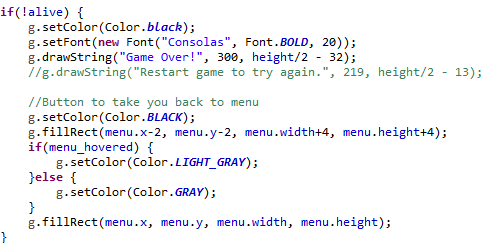
Now for collision. I’ve created a life system based on the difficulty you chose so that if you touch the explosion then you lose a life until you die at 0 lives. The collision with bombs is a bit messy since I first had to go through stage.java to access the explosions array (I’d make this cleaner if I were to do it again) but it works just fine and is pretty good on memory as well

First, since if a bomb in the array is null, then all the bombs after it are null (mentioned earlier, bombs are created in order). This means I can save a lot of processing power by breaking out of the for loop if I hit a null bomb. The next bit is the collision. First the algorithm checks to see if the bomb is actually damaging. This again saves a bit of processing power as bombs that have already exploded don’t have to be checked. It will check through all the explosion particles in said bomb and if the player is touching one, it will remove a life and activate temporary invulnerability which is handled here

Here’s a gif of the collision and invincibility in action : https://gyazo.com/85b1a1304919f120666198ba5cd97a46

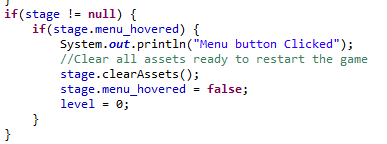
Creating a button that will bring the player back to the menu

Essentially, when the player dies, I have a button in the middle of the screen drawn which will take the player back to the main menu drawn here:

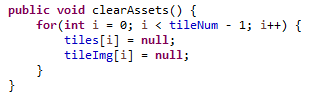


The clicking of the button is detected by the GamePanel itself so that everything can be reset, so when it is clicked, the level is set to 0 and memory heavy assets (basically just the tiles) are removed to save on space.

The click is detected here:



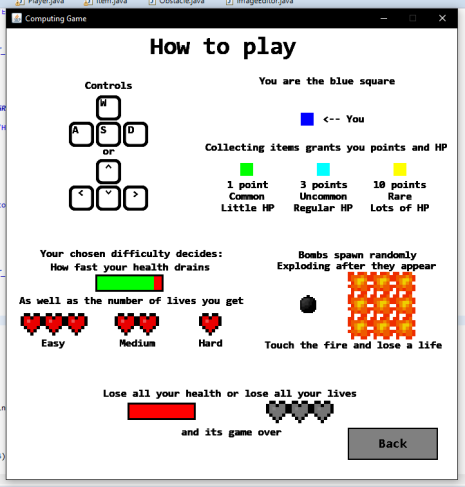
And assets removed here:



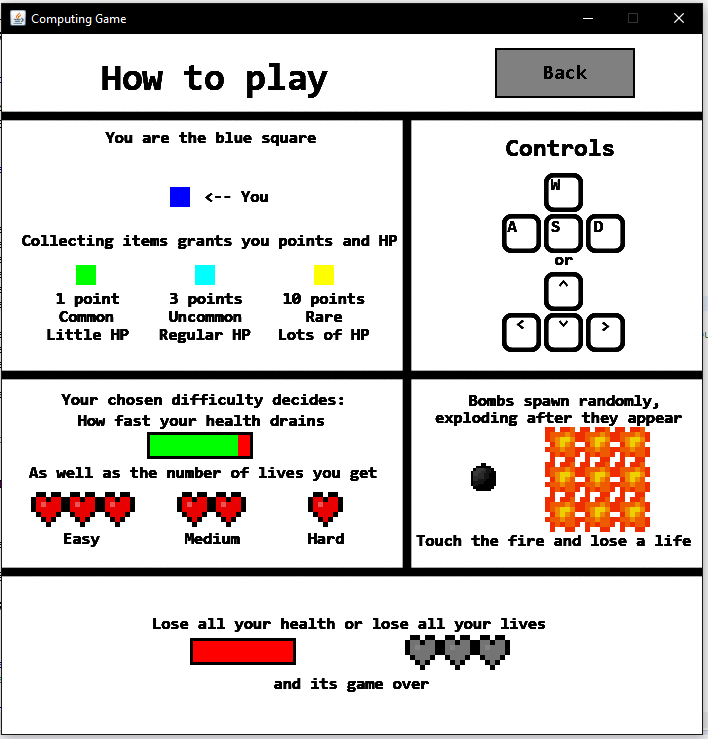
Creating a How to Play Screen

Now that all the mechanics for my game are complete, I can make a How to Play screen.

Since there are quite a few parts to the game and my target market might not fully understand them, I decided to make a small visual “How to play” screen to make sure anyone that plays knows how the game works. First was designing the screen. For this I came up with a first design and gave it to a few friends to look at. This is what it looked like



The general consensus was that the information was good and well presented but it was hard to read because everything is just kinda jumbled together. One friend suggested separating each section off with lines to make it easier to read so I gave that a go which can be seen below



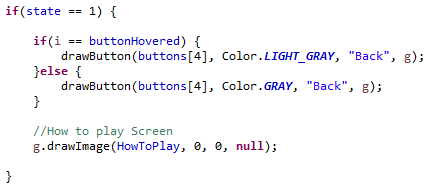
This is the second design I came up with. Its much easier to read and looks a lot more organised than the first design. This may go through more design changes in the future but for now I think this is a suitable How to Play page.

Programming in the How to Play Page

So before this, there was no system for switching menu pages as there was only 1 menu page. Because of this, I had to make a state variable that indicated what part of the menu the user was in.



Now I have this, I can create a How to Play button and have it so that when you click it, it switches the state to 1. Then I have it so that the stage buttons and the how to play button are no longer drawn but the back button is which is decided by a simple if statement in my display() method



Heres a gif of it working as intended : <https://gyazo.com/38dd6a85ccd4aae66c934af60fd7f013>

**Known Bugs**

1. Player will clip through blocks if they enter diagonally on the exact pixel of the corner of the block

<https://gyazo.com/47719e135d6cf1d1be0c0ba9044f03fe>

Priority : Low

During actual gameplay its very rare for this to happen and in my map there are not any places to where this would have to player go outside the map or otherwise. If a block is entered, hit detection still works as intended on the surrounding blocks.