The Legend of the Parallax

Developers and Programmers: (Jacob Goldstein && Patrick Cook)

Objective

- As AB Computer Science students, our semester project was to develop an application or game with a focus on GUIs. The purpose was to allow us to teach ourselves about more in depth areas of Java.

Concepts

- For the first couple of day we took a lot of time to develop a good foundation for our game. It didn't take us too long to realize that 3D is way above our level of knowledge so we decided to build a top-down 2D game. We wanted to create a game that felt and looked like one of the older Legend of Zelda games.

Where to start?

- As you guys will learn, developing your idea before coding it will drastically improve organization and efficiency. Instead of just jumping into a project without the advanced knowledge required to make the game, we decided to study up. For the first week or so we read a lot of the basic setups for creating an application that can “hold” our game. Once we though about the main classes/objects that our game would require we began to start coding.

Movement

- Throughout the first month we focused on creating a basic map that could be traversed using a character. This included using WASD to control your character. Here comes our first major problem! Our link character wouldn't move even though our code was correct. Solution: Trash BlueJ and move on to a better IDE such as Net Beans. Literally all we had to do was copy all of our classes into a new Net Beans project and guess what? It worked, who knows why BlueJ didn't work? There goes 2-3 hours of frustration for nothing!

Collisions

- This is the core for practically every game you'll ever play! Ironically this took us forever to successfully implement into our game. The concept is quite simple though. Our game was based on tiles, which means that a tile is either passable or not passable. So this is where all those 2D arrays you guys hate come into play! For every tile on our map we put either a 0 (passable) or a 1 (blocked) in a 2D array. Add some math and a few lines of code and you have collisions!

```
collisionMap = new int[][]{
    //0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //0
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //1
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //2
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //3
    {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1}, //4
    {1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1}, //5
    {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1}, //6
    {0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //7
    {0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1}, //8
    {0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //9
    {0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1}, //10
    {0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1}
};
```

What's a game without enemies?!

- Our team's lead artist, Jacob, designed our terrifying enemies! Everything from pages upon pages of concept art to prototype clay models to creating drafts in Photoshop! Basically we randomly drew enemies in Photoshop!

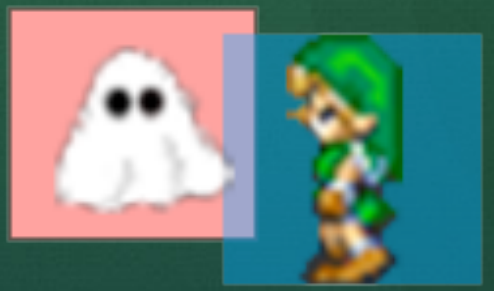
Collisions For Objects

- Once the map had collisions so that Link couldn't drown in the water, we could move on to other collisions. We needed Link to lose a life if he touched an enemy. To do this we created a rectangle the size of Link and then another box for each enemy. Java's Rectangle class has a built-in method called intersects(Rectangle rect)which can be used to check for collisions. It looks something like this:

- if (linkRect.intersects(enemyRect)){

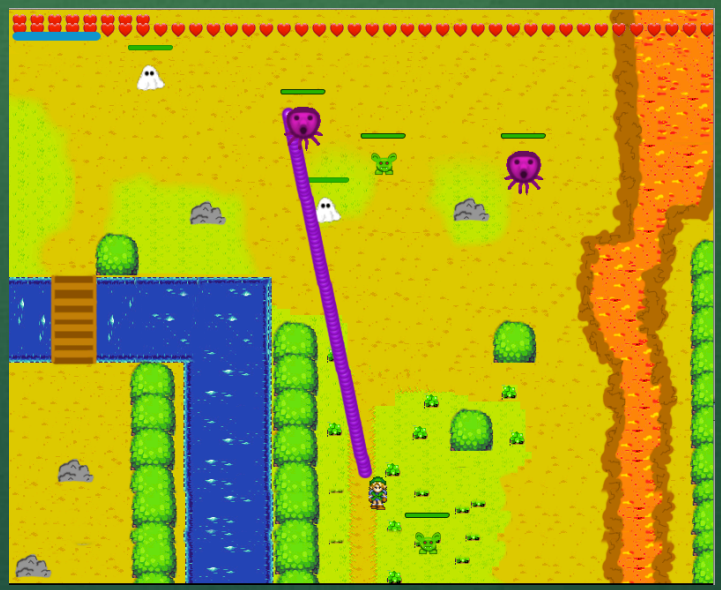
  //take life away

  }



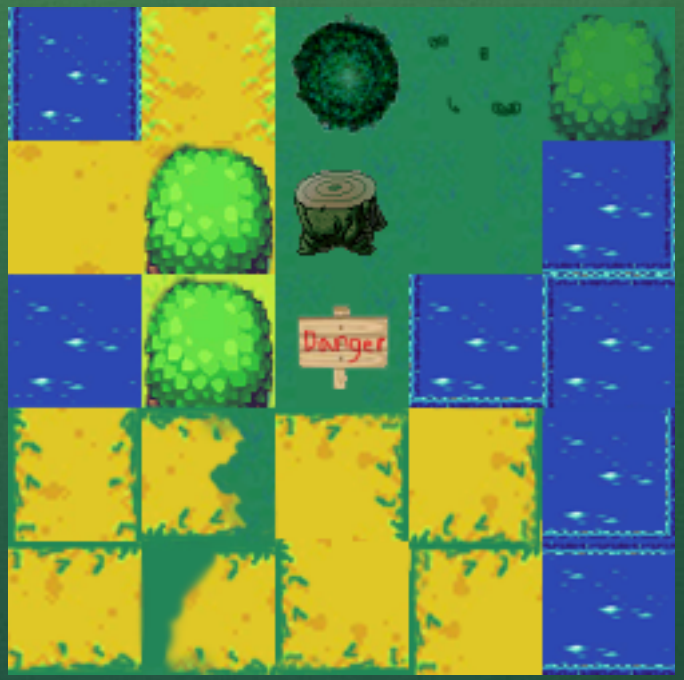
Enemies, Collisions, Now What?

- As many of you know we hit a point where we had developed a slightly entertaining version of our game. We let some of you beta test it and most of the time the game had huge problems. We quickly attended to these. Some of which were that the arrows could be redirected once shot by pressing space again. Another was that the enemies shot hundreds of bullets a second!



Map Creation

- To make our tiles maps we made a sprite sheet and then used an application called Tiled to create our maps. This made it so that each world had a different map.



Finishing Touches

- Our first world is beaten when the player successfully kills all enemies. We decided that adding more enemies each world wasn't adding to the game. Instead we decided to redesign World 2. We made the world have a boss that is actually difficult to kill. We added a chest and a key that is required to pass through a blocked area. This allows you to beat the level. We also added music that changes depending on which world you are on. There are sound effects for dying and winning, opening chests and for hitting enemies. We also added health bars for each enemy! This is helpful because of the thousands of beta testers we had, many were unsure of how much health each enemy had.

Snippets of code that make or break our game

- Removing an object from an ArrayList is actually possible while in a for-each loop!

```
public void arrowCollision(Graphics2D g2d) {

    //create list of enemies and arrows to remove
    HashSet arrowRemove = new HashSet();
    HashSet enemyRemove = new HashSet();
    for (Arrow c : quiver) {
        for (Enemy x : enemies) {
            if (c != null) {
                //does the arrow interect any enemies?
                Rectangle e = new Rectangle(x.getX(), x.getY(), x.getWidth(), x.getHeight());
                Rectangle p = new Rectangle(c.getX(), c.getY(), c.getWidth(), c.getHeight());

                if (p.intersects(e) && !(x instanceof BossOne)) {
                    s.hit();
                    x.setHealth(x.getHealth() - 1);
                    expAnim = new Animation(x.getDeathAnim(),
                            134, 135, 4, 45,
                            false, x.getX() - 50, x.getY() - 50, 0);
                    expAnim.setTimer(30);
                    arrowRemove.add(c);
                    if (x.getHealth() <= 0) {
                        enemyRemove.add(x);
                    }
                }
                if (p.intersects(e) && x instanceof BossOne) {
                    s.hit();
                    x.setHealth(x.getHealth() - 1);
                    arrowRemove.add(c);
                    if (x.getHealth() <= 0) {
                        new Chest().poofAnim.Draw(g2d, 0, 0);
                        drawChest = true;
                        enemyRemove.add(x);
                    }
                }
            }
        }
    }
    if (expAnim != null) {
        expAnim.Draw(g2d, 0, 0);
    }
    quiver.removeAll(arrowRemove);
    enemies.removeAll(enemyRemove);

}
```

Cont.

- This is our maps collision detection!
- It checks our array for either a 1 (blocked) or a 0 (open).

```
public void setOpenAndBlocked() {
    blocked.clear();
    for (int c = 0; c < collisionMap.length; c++) {
        for (int d = 0; d < collisionMap[0].length; d++) {
            if (collisionMap[c][d] == 1) {
                blocked.add(new Rectangle(d * 50, (c) * 50, 50, 50));
            }

        }
    }
    open.clear();
    for (int c = 0; c < collisionMap.length; c++) {
        for (int d = 0; d < collisionMap[0].length; d++) {
            if (collisionMap[c][d] == 0) {
                open.add(new Rectangle((d * 50)+10, (c*50)+10, 10, 10));
            }
        }
    }
}

public boolean blockedMove(int x, int y, int width, int height) {
    Rectangle p = new Rectangle(x, y, width, height);
    for (Rectangle r : blocked) {
        if (p.intersects(r)) {
            return true;
        }
    }

    return false;
}
```

Animations

- Creating animations proved to be one of the most difficult parts. Our animations works by painting an image from a sprite sheet, and then after a certain amount of time it paints the next image in the sprite sheet.

```
public void Draw(Graphics2D g2d, int dirX, int dirY) {
    x += dirX;
    y += dirY;
    this.Update();
    if (timer > 0) {
        // Checks if show delay is over.

        if (this.timeOfAnimationCration + this.showDelay <= System.currentTimeMillis()) {
            g2d.drawImage(animImage, x, y, x + frameWidth, y + frameHeight, startingXOfFrameInImage
        }
        timer--;
    }
}
```

So in the end, what did we learn?

- Way, way too much to include in a presentation. In a nutshell we understand the basic principles of making a 2D game using a GUI. We learned how to tweak our game to run efficiently. We now understand how the fundamentals of most 2D games work. We also have come to the realization that the games we play have tons of time put into them. Next time you play a game, take a second to realize how much time was put into the development of it. The most valuable thing we have learned is that simple coding is always better. Most people in this classroom have asked us how many lines of code we've coded. Most think that the more the better, however, this is not true. The most efficient programs run using the least amount of code possible. Just think, our game has to calculate all the collisions, calculate movement for the enemies and player, play music and run tons of lines of code. This is all repeated every 16 milliseconds for 60 fps. For those wondering, our game has a total of 2296 lines of code. However, if we remade our game, we could make it with less than 1500.