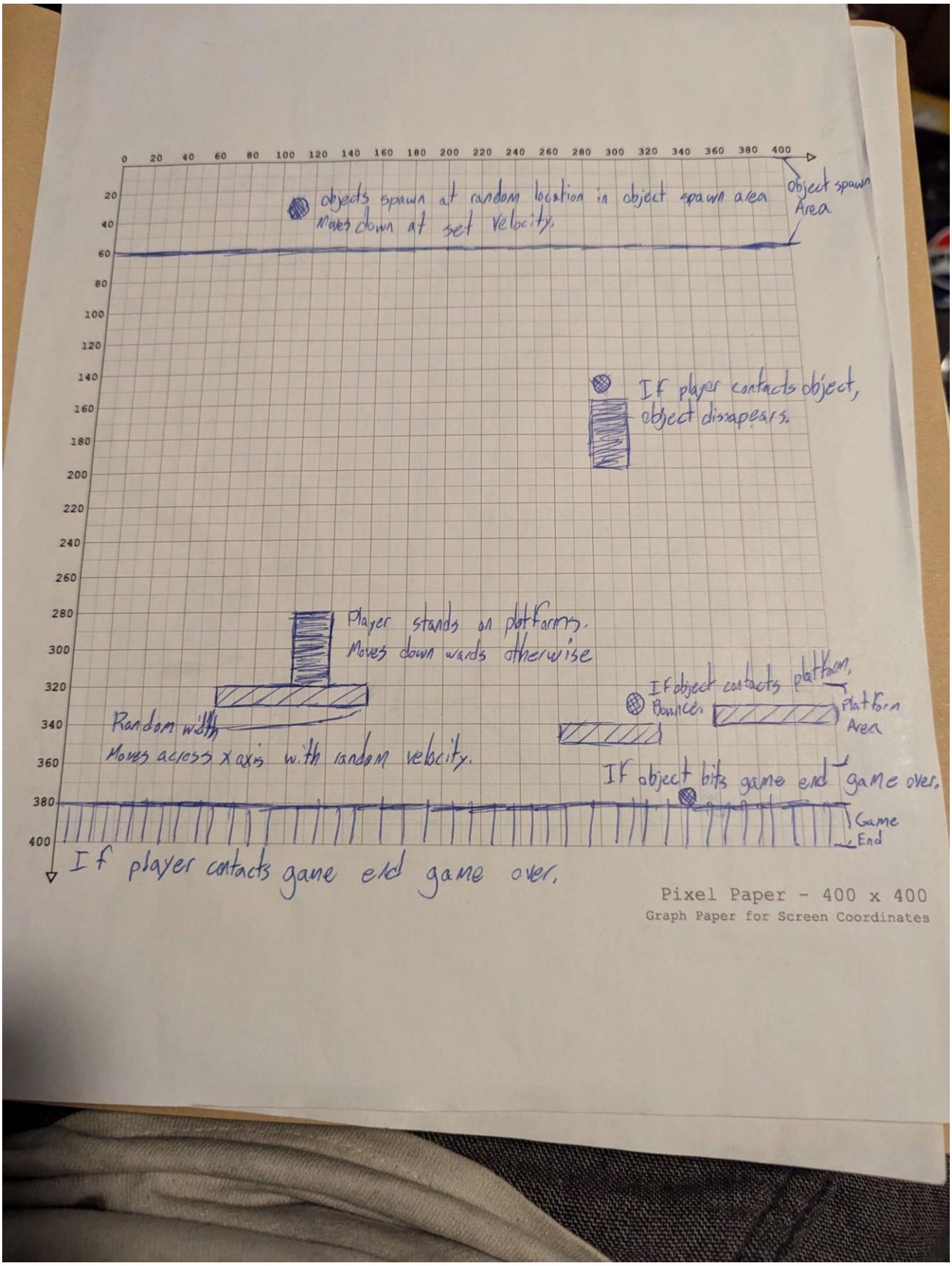
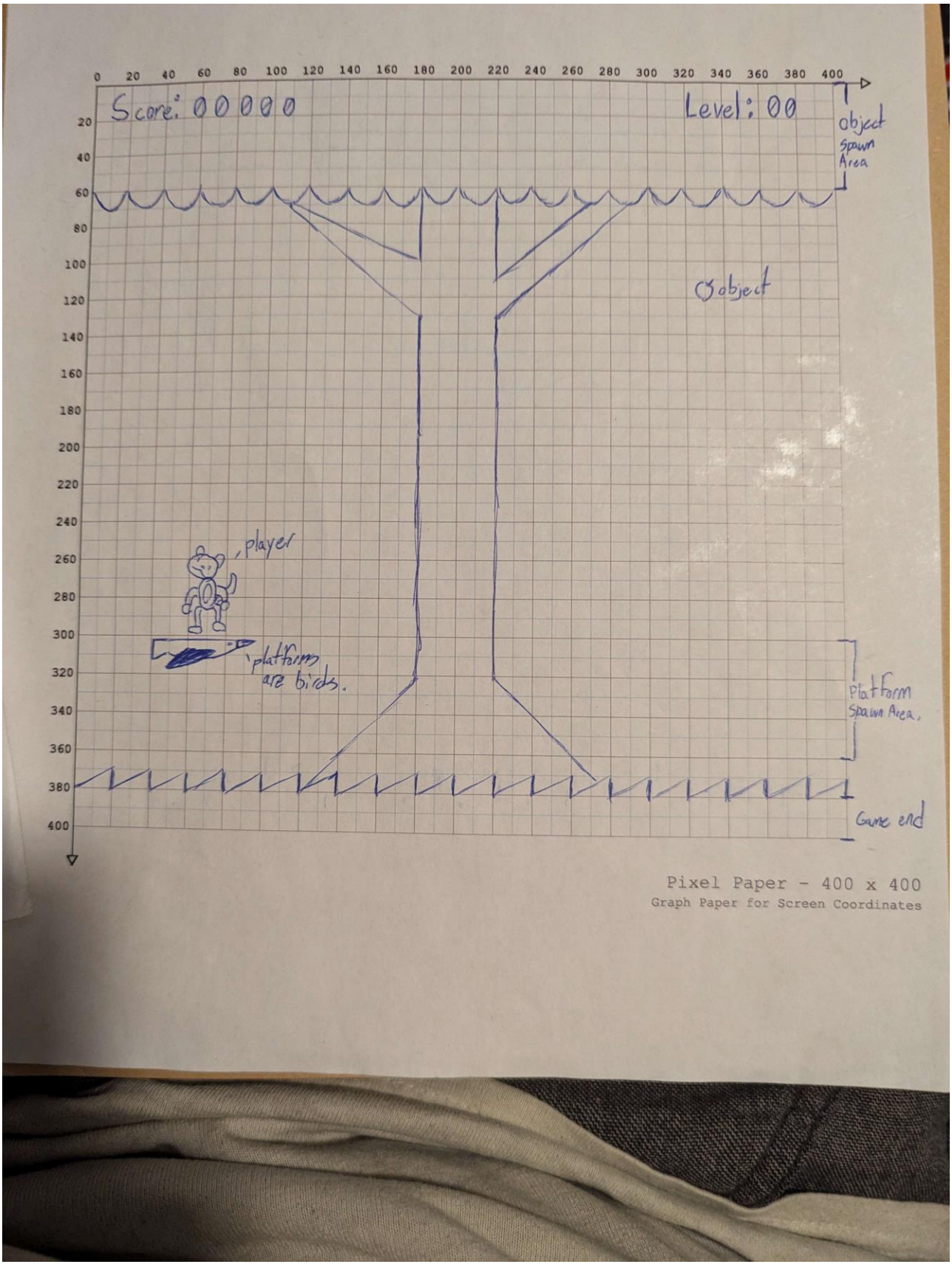


Pixel Paper - 400 x 400

Graph Paper for Screen Coordinates



Pixel Paper - 400 x 400

Graph Paper for Screen Coordinates

Game Dev Fonadations Assignment 3 2D GAME PROJECT E.Muller

What kind of game? Needs collision. Could clone, don't want to.

Concepts required: 1. Encapsulation. 2. Array of a class type. 3. Use methods of motion to create a dynamic game.
4. Use methods of collision detection to create a highly interactive game.

Moving platforms could be done using an Array. Would be a great example of collision too.
A dynamic object the player could interact with?
Volley ball? Bullets? Seesaw? Button?

Vectors and time!
Maybe vectors for platform idea?

Player could have a tool?
Learn how to implement sounds.

side-Scrolling or singular level?

Maybe make a point system.

Try making a single-player game for now.

Give platforms a variable x movement. Would make moving platforms.
Maybe some small objects to carry in a bowl? ~~Bal~~ Balancing?

Player could catch things.

Game ideas?

Catch fruit game?
Player jumps between platforms trying to catch fruits.
Different fruit have different scores. Player reaches lose condition if they miss three fruit in a row, or fall between the platforms.
Player reaches new level after 60 seconds. Platforms move faster every level.

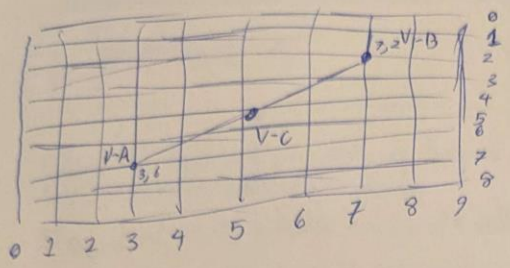
Platforms are an object. Character is an object.
Fruit are objects. Fruit game is a go.
Character is monkey!

Discussed collision with prof. A function acting as a bool can assist with platform ~~collising~~ Collision. This can make sure the monkey can only jump when in contact with the platforms!

example: bool IsCollidedWithPlatform()
IsMonkeyFeet >= PlatformTop && monkeyLeft >= Platform ~~Left~~ Right && MonkeyRight < PlatformLeft
{
~~Monkey Jump~~
}
Return: True

If You get the idea. A true value prevents jumping. A false value allows it. I think.
Can't test until I figure out the ~~jump~~ movement and collision mechanics anyway.

I need to understand how I can do movement. The prof says Int or float methods are fine, but I really should figure out vectors. It will allow for time commands. A vector is one point. Vector 2 is called that because it holds an x & y position.



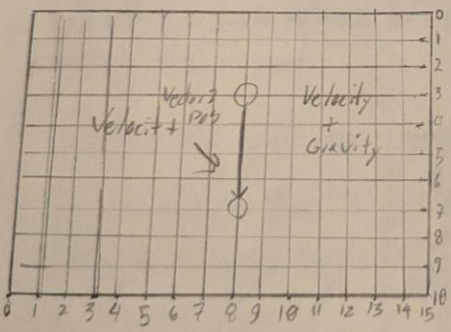
With Vector 2's Lines are easier Also, finding point between them are easier. I need to think of these as mathmatic functions. The hard part is using them together.

Encapsulating the code for each unique type of object is needed. I'll create one for the platform, fruit & player object.

Back to Vectors. If I use Vector 2 and Time functions together, I should get the result needed I'm pretty sure this was done in class, but code theft is pretty evil.

Let's try one object for now. A funny circle.

Just got some new supplies!



I've looked over module 4 again to better understand Vectors. They give a really basic set of code for gravity.

```
Vector2 gravityForce = gravity * Time.DeltaTime;
velocity += gravityForce;
position += velocity;
```

```
Setup: Vector2 position;
       Vector2 velocity;
       Vector2 gravity = new Vector(0, +10);
```

This is the ↓ down force

↑ Ted is slow. Changing to 30

Velocity is effected by gravity. Position then is modified by velocity.

To ensure sliding does not occur, I could use an "If statement."

```
if (IsObjectCollided = false && Velocity < 1)
  {
  ObjectVelocity = 0;
  }
```

Collision for fruit will be simple as long as I make them circles. That lets me do some easy math; Vectors add more simplicity.

If any object touches a fruits radius it's colliding of course so:

```
Vector2 fruitDistance = playerPosition - fruitPosition;
if (fruitDistance <= fruitRadius)
  {
  FruitCollected = True;
  }
```

fruitRadius likely won't be a vector.
Fruit collected will lead to a while statement made for object collection.

Back to platform movement for a second.
If I want the platform to move sideways, I can use the gravity code on the x axis. In order to have platforms continually returning from the sides, I can use an If statement that swaps between left and right versions of the code depending on how far the platform went.

```
if (platformPosition.X <= -30)
  {
  IsGoingLeft = true;
  }
if (platformPosition.X >= 830)
  {
  IsGoingLeft = false;
  }
```

```
if (IsGoingLeft)
  {
  Vector2 directionalMovement = platformDirection * Time.DeltaTime;
  PlatformSpeed += directionalMovement;
  platformPosition += PlatformSpeed;
  }
else
  {
  Vector2 directionalMovement = platformDirection * Time.DeltaTime;
  platformSpeed -= directionalMovement;
  platformPosition += platformSpeed;}
```