AP Computer Science A - Project Proposal

Game: **"Triggering Insanity"**

Kevin Kwan

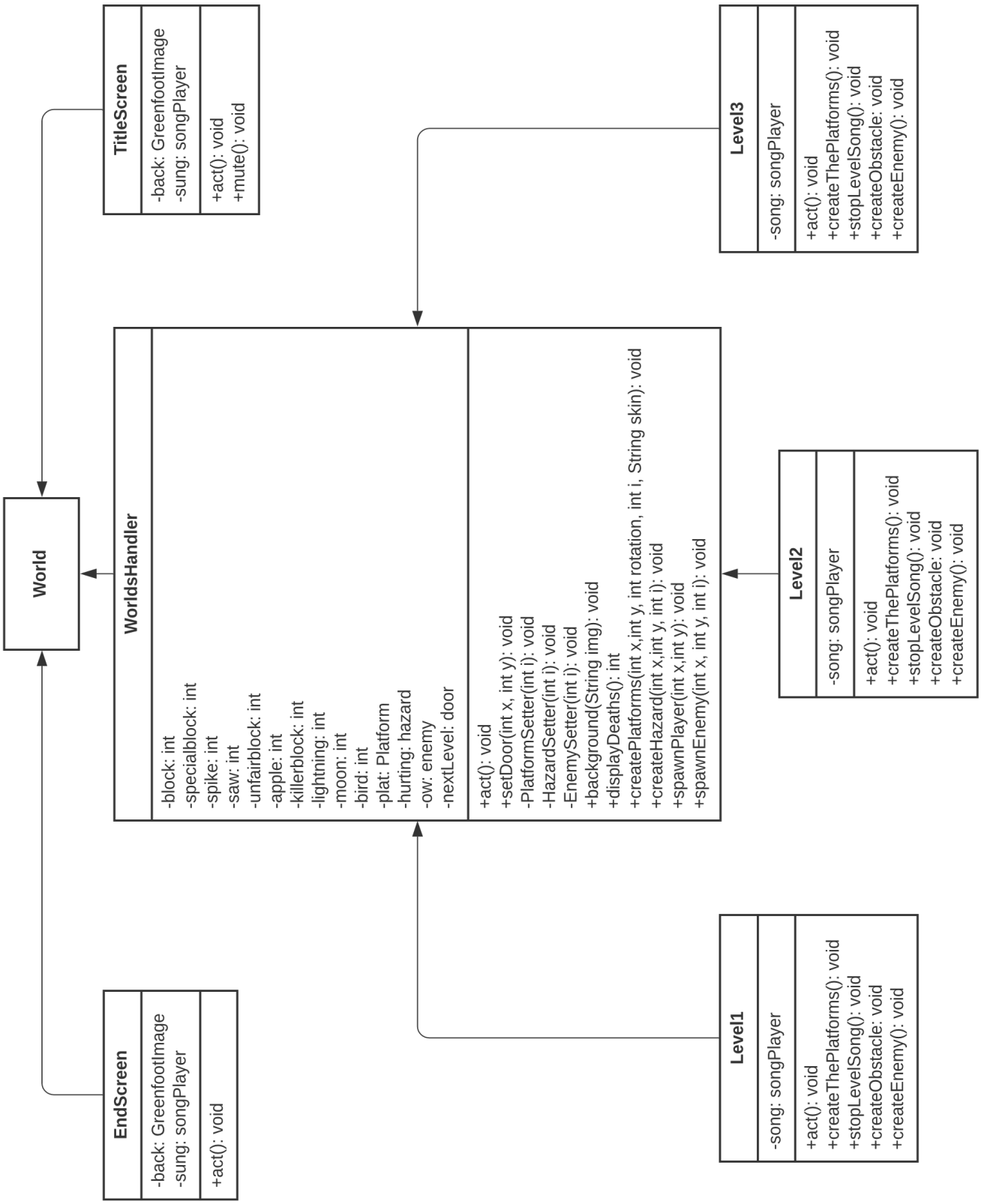
Project Summary:

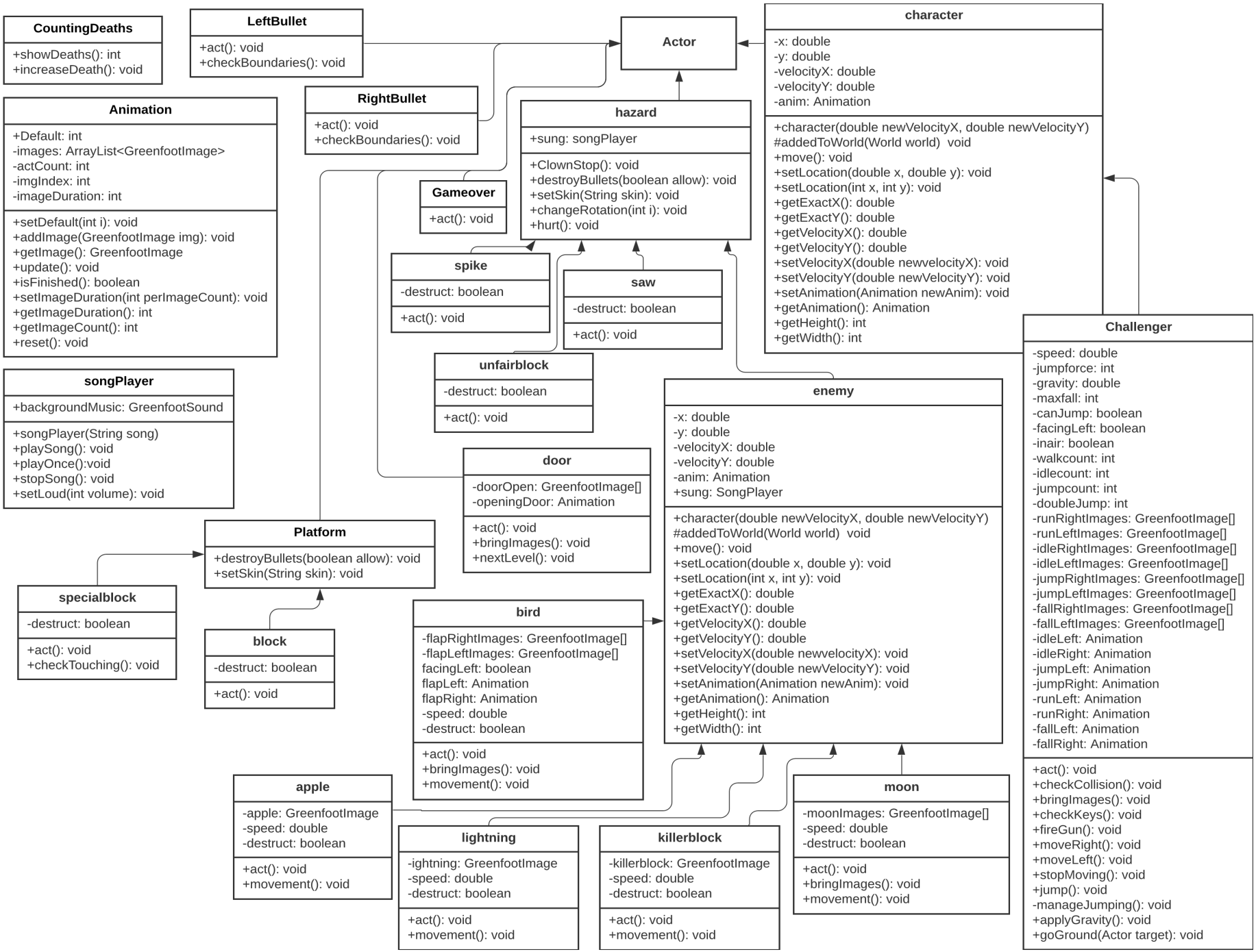
My game, **"Triggering Insanity,"** is going to be an extremely difficult, single-player, 2-D platform game where the player has to traverse many levels filled with hazards to reach the goal of getting "100%", which is an object in the game. 100% is also a pun for their grade and 100% completion of the game. However, this game isn't like any Mario platformer that provides entertainment. This game is built to induce rage within the player as the levels are filled with unforeseeable traps, random, unfair events, intentionally poor hitboxes, and very hard platforming. Invisible spikes will appear out of nowhere, seemingly "harmless" things will become deadly, and precise gaps will force the player to restart fifty times before they get past an obstacle. After playing for a while, the player's mind will deteriorate and be filled with so much anger that this game will haunt them for the rest of their lives.

The player will be controlling the main character, a little boy in a superhero costume, using the arrow keys/WASD to move and jump. Double-tapping [W] also allows for double-jumping. The character will be animated using sprites. The character will also have a gun to shoot potential enemies and blocks which can be activated using [space] on the keyboard. The player, whenever they die or want to restart, can press [R] on the keyboard to restart the level. Platforms will be made in a world with different kinds of blocks depending on the level along with generated hazards. Hazards include saws, spikes, blocks, enemies, invisible versions of the aforementioned objects, and more. Touching any of these hazards will leave the player dead in one-hit, and the level that they are playing on will restart. The player will have to beat the level by platforming and getting past all of the obstacles without dying and reaching a doorway to go to the next level. The level design will change drastically every time the player advances a level and may introduce new surprises. While there aren't any rules, the player can use any technique or strategy to control the character and get to the exit. The enemies will not have artificial intelligence, as the game should already be hard enough. Deep learning and neural networks will not be employed to make the game unbeatable. Instead, enemies will be coded to follow the player or move on a set path. When the player touches an enemy, they will die and the level will restart. Some enemies can be killed using the main character's bullets.

While the concept of game may seem simple and common due to the variety of platforming games, the challenging part of this game is its most appealing aspect. Most platformers, notably the Super Mario Bros. franchise, are made for children to play, enjoy, and have a good time. Those games are easy to play and beat. However, my game is the opposite. The player must play with precision, patience, and intelligence in order to best my game with simple controls and a straightforward objective. It will be an accomplishment to beat such a challenging game.

UML Diagram: (Split into 2 pages due to large size)




CountingDeaths
+showDeaths(): int
+increaseDeath(): void
LeftBullet
+act(): void
+checkBoundaries(): void
Actor
character
-x: double
-y: double
-velocityX: double
-velocityY: double
-anim: Animation
+character(double newVelocityX, double newVelocityY)
#addedToWorld(World world) void
+move(): void
+setLocation(double x, double y): void
+setLocation(int x, int y): void
+getExactX(): double
+getExactY(): double
+getVelocityX(): double
+getVelocityY(): double
+setVelocityX(double newvelocityX): void
+setVelocityY(double newVelocityY): void
+setAnimation(Animation newAnim): void
+getAnimation(): Animation
+getHeight(): int
+getWidth(): int
Animation
+Default: int
-images: ArrayList<GreenfootImage>
-actCount: int
-imgIndex: int
-imageDuration: int
+setDefault(int i): void
+addImage(GreenfootImage img): void
+getImage(): GreenfootImage
+update(): void
+isFinished(): boolean
+setImageDuration(int perImageCount): void
+getImageDuration(): int
+getImageCount(): int
+reset(): void
RightBullet
+act(): void
+checkBoundaries(): void
hazard
+sung: songPlayer
+ClownStop(): void
+destroyBullets(boolean allow): void
+setSkin(String skin): void
+changeRotation(int i): void
+hurt(): void
Gameover
+act(): void
spike
-destruct: boolean
+act(): void
saw
-destruct: boolean
+act(): void
unfairblock
-destruct: boolean
+act(): void
Challenger
-speed: double
-jumpforce: int
-gravity: double
-maxfall: int
-canJump: boolean
-facingLeft: boolean
-inair: boolean
-walkcount: int
-idlecount: int
-jumpcount: int
-doubleJump: int
-runRightImages: GreenfootImage[]
-runLeftImages: GreenfootImage[]
-idleRightImages: GreenfootImage[]
-idleLeftImages: GreenfootImage[]
-jumpRightImages: GreenfootImage[]
-jumpLeftImages: GreenfootImage[]
-fallRightImages: GreenfootImage[]
-fallLeftImages: GreenfootImage[]
-idleLeft: Animation
-idleRight: Animation
-jumpLeft: Animation
-jumpRight: Animation
-runLeft: Animation
-runRight: Animation
-fallLeft: Animation
-fallRight: Animation
+act(): void
+checkCollision(): void
+bringImages(): void
+checkKeys(): void
+fireGun(): void
+moveRight(): void
+moveLeft(): void
+stopMoving(): void
+jump(): void
-manageJumping(): void
+applyGravity(): void
+goGround(Actor target): void
songPlayer
+backgroundMusic: GreenfootSound
+songPlayer(String song)
+playSong(): void
+playOnce():void
+stopSong(): void
+setLoud(int volume): void
door
-doorOpen: GreenfootImage[]
-openingDoor: Animation
+act(): void
+bringImages(): void
+nextLevel(): void
enemy
-x: double
-y: double
-velocityX: double
-velocityY: double
-anim: Animation
+sung: SongPlayer
+character(double newVelocityX, double newVelocityY)
#addedToWorld(World world) void
+move(): void
+setLocation(double x, double y): void
+setLocation(int x, int y): void
+getExactX(): double
+getExactY(): double
+getVelocityX(): double
+getVelocityY(): double
+setVelocityX(double newvelocityX): void
+setVelocityY(double newVelocityY): void
+setAnimation(Animation newAnim): void
+getAnimation(): Animation
+getHeight(): int
+getWidth(): int
Platform
+destroyBullets(boolean allow): void
+setSkin(String skin): void
specialblock
-destruct: boolean
+act(): void
+checkTouching(): void
block
-destruct: boolean
+act(): void
bird
-flapRightImages: GreenfootImage[]
-flapLeftImages: GreenfootImage[]
facingLeft: boolean
flapLeft: Animation
flapRight: Animation
-speed: double
-destruct: boolean
+act(): void
+bringImages(): void
+movement(): void
apple
-apple: GreenfootImage
-speed: double
-destruct: boolean
+act(): void
+movement(): void
lightning
-ightning: GreenfootImage
-speed: double
-destruct: boolean
+act(): void
+movement(): void
killerblock
-killerblock: GreenfootImage
-speed: double
-destruct: boolean
+act(): void
+movement(): void
moon
-moonImages: GreenfootImage[]
-speed: double
-destruct: boolean
+act(): void
+bringImages(): void
+movement(): void

Screenshots:

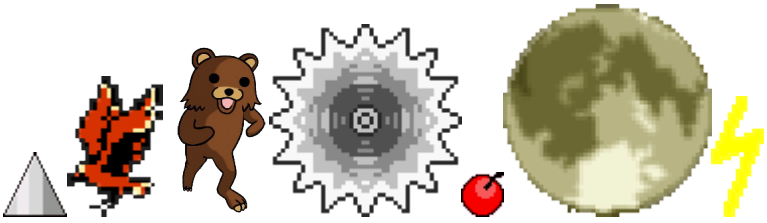
This is the start screen that players will see when starting up the game.

These are the sprites that make the main character walk/run.

These are the sprites that make the main character jump and fall.

These are the sprites that will make up the character's idle animation.

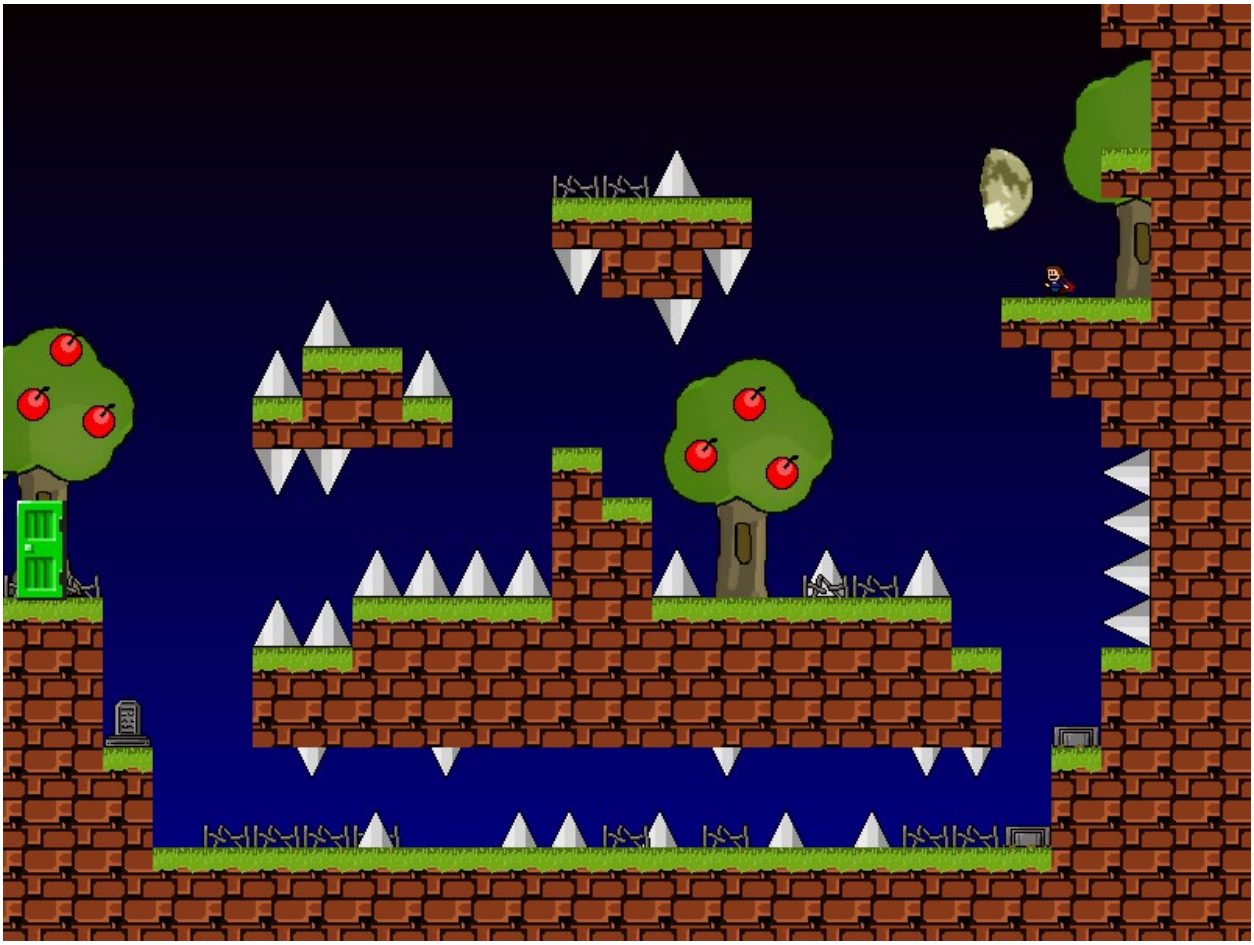
These are some of the potential hazards that can kill the player in one-hit.



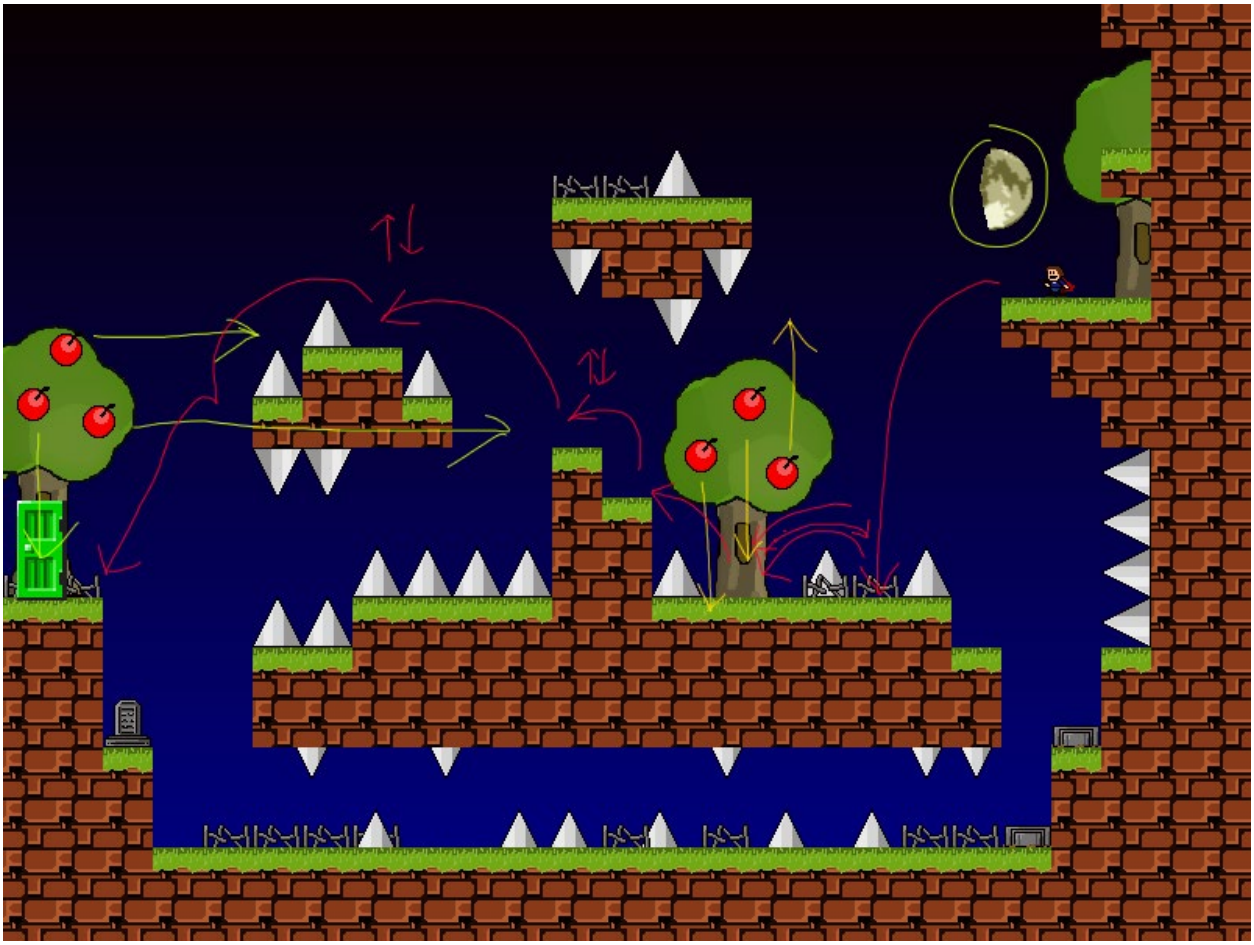
These are some of the platform blocks that the player can jump on and interact with.

(More screenshots below)

This is an example of a potentially difficult level. Whenever the player beats a level, the screen changes to a new level where the player can continue to struggle until that level is beaten. Levels are usually filled with traps such as spikes, invisible blocks, and innocent things like the moon and apples which can kill the player. Levels are built to be unfair and use the player's lack of knowledge to its advantage.



Here is an example of how to navigate through a level like this. The directions of the hazards are marked with yellow and the ideal path that the player should take is marked in red. These markers will not be in the game in order to make the player decide on which path to choose and experience the surprise/unfair factor of obstacles. These markers are provided to show what is going on in this screenshot.



This is what a player will see when they are killed by something. A “Game Over” overlay will appear, the player will have the option of restarting the level. When the player dies, the character’s body will explode into body parts and blood will spill onto the ground.