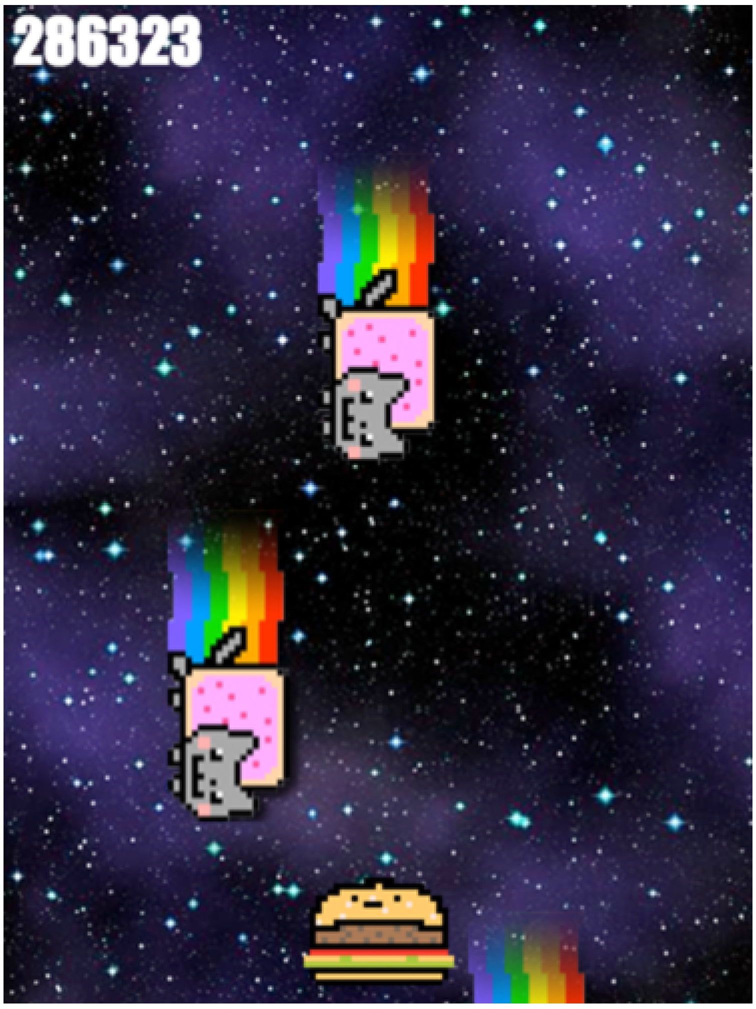
Object Oriented Programming Game Project



Introduction

This project was designed as a way for you to practice and learn about Object Oriented Programming. A good bit of code has already been provided for you, and it makes for an *almost* working game.

The project is divided in two parts. In a first part, you will fix and implement some basic functionality. In the second part, you will be able to customize the game as you wish!

Because there is a fair bit of JavaScript in this app, we have split code into more manageable pieces. You wil find all of the code in the `js` folder.

Because we split the code into many files, we need to add the `<script>` tags in the right order in the `index.html` file. This is already done for you, but it's important not to change the order.

The Game

This game is called **I CAN HAZ CHEEZBURGER?!??**. In the game, you play as an anthropomorphic cheeseburger. The only thing you can do is move left or right with the arrows of your keyboard.

The goal of the game is to stay alive as long as possible by avoiding the Nyan Cats who are raining from the sky and trying to *haz* you. The longer you stay alive, the higher your score!

At the moment, the game is a bit broken. This project will have you first fix the broken game, then add your own features to it. 😃

Instructions

1. Open the `index.html` file in your browser, and observe what goes on when you load it.
2. After reading the instructions of the project, **take a look at the provided code**. Don't worry if you don't understand *everything*, but try to get a general feel for what the code is doing. There are extensive comments throughout the code.
3. Once the provided code has been consulted, go through the Assignment section at your own pace.

Assignment

1. Let's make the game actually end

If a Nyan Cat gets to you and eats you, the game keeps going on. Let's fix this!

First, look at the `gameLoop` method of the `Engine`. There's a part of the function that calls `this.isPlayerDead()` to verify if the player has died based on the current situation.

Next, look at the `isPlayerDead` method of the `Engine`. Notice that it's always returning `false`, which means that the player is always reported to be alive.

Here, we are going to rewrite the code of this function to actually check if the player should be dead. We will do this by looping over all the enemies, and checking if their box overlaps the player box.

If at least one enemy overlaps the player, then your function should return `true`. Otherwise it should return `false.

A good strategy would be to `console.log` both `this.player` and `this.enemies`. When you look in the console, you will see that those two objects contain the information necessary to detect a collision.

> *HINT:* In Javascript, the 0,0 position is the *top left* corner. This means that if an item is at `{x: 0, y: 0}`, it's in the top left, not the bottom left. As `y` values increase, items move down.

> *HINT:* A collision takes place when one of the enemy's boxes overlaps the players. All enemies and players are positioned with `x` and `y` coordinates. There are also helpful constants like `ENEMY_HEIGHT` and `PLAYER_HEIGHT` you can use.

> *HINT:* Try calling `console.log` on the player. You might notice that you don't have all the information you need. Feel free to modify `Player.js` to add some more info to `this`.

2. Flavor it!

Having completed the basic section of this project is already great! However, next up is the fun part: customizing and evolving the game. Since this is an open-ended activity, we will give you some suggestions. Feel free to use them or not.

Minor changes:

- New player character.
- New enemies.
- New gameboard.
- Background music.
- Sound effects.
- Change the size of the gameboard.
- A restart button when the game is over.
- A start button.
- Animated charaters.
- Let the player to also move up/down.
- Make enemies shoot from the sides of the screen too.

Major changes:

- Add a lives system.
- Add a score system.
- Increase the difficulty level of the game as time passes by making the enemies go faster and/or increasing the number of enemies.
- Allow the player to shoot bullets at the enemies.
- Add another type of entity called `Bonus` that the player can pick up for bonus points.
- Add power ups to the game.

> **Don't stop!** These suggestions are only here to get you started 😃

Project Success Guidelines

In order for the project to be deemed *successful* and for you to receive a passing grade

- you need to do step 1 completely.
- add at least 1 major change.
- add at least 1 minor change.

Submitting the Project

When submitting the pull request for your project, you must include comments that explain:

- What minor changes were added to your game.
- What major changes were added to your game.
- A warning if there are sounds (music, sound effects, etc.)

Screenshots are optional, but nice to have.

Useful Links

- https://javascript.info/keyboard-events

- https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
- https://www.w3schools.com/jsref/met_element_getboundingclientrect.asp
- https://developer.mozilla.org/en-US/docs/Web/API/Element/getBoundingClientRect

Useful Terms

> For those of you who are not familiar with games and/or their terminology.

- **Engine**: games usually run off a `game engine`, an environment that powers the game. In this project it's represented by the `Engine.js` file.
- **Hitbox**: the physical space that an entity occupies in the game. It is usually `rectangular` in shape and `detects collosions` on it's borders.
- **Gameplay**: the way a game is played. This includes the rules and other factors that affect how a player can play the game.
- **Gameboard**: the area on screen where the game is played.