General Assembly

Project 1 - (Rick and Morty)'s Frogger

R&M's Frogger is a game that starts as a game based on Frogger. Our frog must walk from the bottom of the grid to the top and reach the portal to escape the aliens that have invaded his dreams. The portal is not the end though, far from it! Our frog must escape his nightmares by travelling back to where they started and putting on their power armour to defeat the evil frogs who invaded his imagination with their froggy nonsense.

Game link

Overview and concept

Technologies used

HTML
- Header including logo and updateable game name 'banner'
- Body containing the 10 x 10 grid for 100 cells total
- Start button and intro graphic
- Audio element to update the src for in-game sounds
- Footer sections with the level no., no. lives left and frog face avatar (a la Doom)

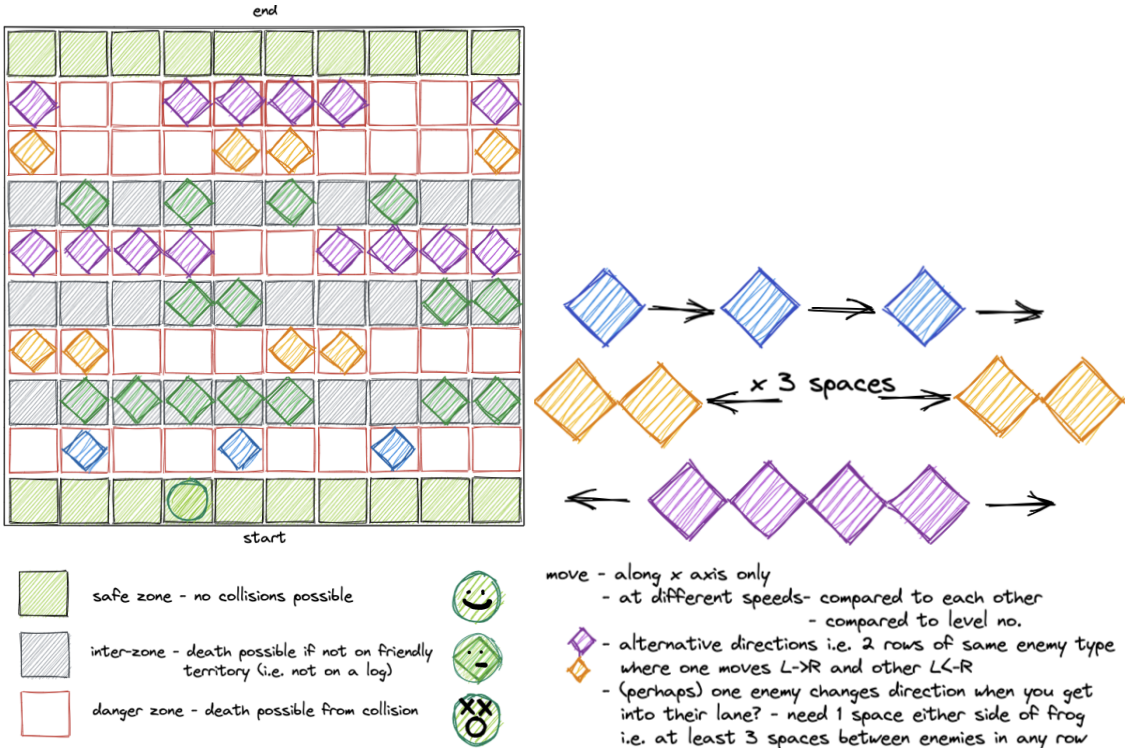
CSS
- Flexbox grid
- Numerous classes that contain the images and gifs used in the grid for enemies, our frog and portals etc.

JavaScript
- Several global variables to store game progression events
- setIntervals to move enemies, delay audio and image changes in response to game conditions and changing of levels. Further intervals set to check for game status and run functions once certain conditions met
- Click event on image used as button to prepare game for startGame function

Approach

I wanted to make frogger as I felt there was a lot of room for creative freedom with the game, so much so that I decided to incorporate mechanics of 2 other games - pac-man and space invaders - in order to make an unpredictable experience for Rick and Morty fans that will hopefully make them laugh. The very nature of the game's theme lends itself well to being unpredictable and absurd(!) Throughout the game, minimal to no instruction is provided deliberately to enhance the mystery of the game. I hope it is obvious what the objectives are but will react to all feedback received from players who try the game. After spending a long time on development, the game is very easy for me to complete as I am very familiar with it.

Enemy types and movement



My initial plan was to have three enemy types, one of which would react to the frog being in its row by changing direction (R>L or L>R). All enemy type would be of different sizes and move at different speeds, becoming increasingly faster the further the player moves ‘up’ the grid. I also planned on having ‘friendly’ lanes of moving ‘objects’ like in the original Frogger game with the logs flowing down the river, but did not pursue the idea after researching how it would be achieved and deciding that it would likely require me to change the game structure so that two different images could be shown in the same cell of the grid.

I started by trying to create the enemy row ‘carousels’ - repeated patterns of movement of the enemies that could scale to larger grids as I believed this would be the biggest challenge to crack and once it was, that the fundamental game mechanic would be in place and everything else could then be addressed. The first enemy set was easy to implement, as they walk across the screen in one direction, with one space between them. A simple checking of the index position of the cell in the grid for presence of the enemy ‘class’ and placing an enemy background image there if not, and removing if present was all that is required.

```
function moveEnemyOne(position, enemyClass) {
  // enemyone default movement L ---> R
  collisionCheck(position)
  if (cells[position].classList.contains(enemyClass)){
    for (let i = position; i < (position + (width)); i++){
      cells[i].classList.remove(enemyClass)
      i++
      cells[i].classList.add(enemyClass)
      collisionCheck(i)
    }
  } else {
    for (let i = position; i < (position + (width - 1)); i++){
      cells[i].classList.add(enemyClass)
      collisionCheck(i)
      i++
      cells[i].classList.remove(enemyClass)
    }
  }
  if (level === 3){
    for (let i = position; i < (position + (width)); i++){
      cells[i].classList.remove(enemyClass)
    }
  }
}
```

The second enemy set was harder to implement as I wanted to have a random spawning of this type of enemy, based on a Math.random multiplier to determine whether or not to spawn an enemy. After researching that I found that it would be quite complicated to insert, so eventually decided on a loop that inputs enemyTwo in a two-(two spaces)-four configuration. Combined with an adjacent enemyOne movement configuration, it provides a reasonable challenge for the player to overcome.

I was determined to keep the grid structure as simple as possible without the use of additional arrays or creating object classes. All movement in the game is represented by the changing of background images on the html elements.

As the game developed further I was less concerned with scaling of the grid size in favour of making the game work as best as it could on a standard 10x10 grid, but the option is there to make the game grid bigger in the future with a few additional lines to create rows dependent on grid size.

Creating the ‘chasing’ mechanic in level 2 for the enemy was surprisingly simple as I did not put any walls up within the grid. The enemy movement is determined by proximity to the frog, first checking if the frog is within 10 index positions +/- of the frogs cell array index in the grid and if not, the enemy moves an entire row closer to the frog.

```
function moveFreddy(){
  if (level === 2) {
    if (currPos > freddyCurPos) {
      if (currPos - freddyCurPos < 10){
        cells[freddyCurPos].classList.remove(freddy)
        freddyCurPos++
        cells[freddyCurPos].classList.add(freddy)
      } else {
        cells[freddyCurPos].classList.remove(freddy)
        freddyCurPos = freddyCurPos + width
        cells[freddyCurPos].classList.add(freddy)
      }
      collisionCheck(freddyCurPos)
    } else if (currPos < freddyCurPos) {
      if (freddyCurPos - currPos < 10){
        cells[freddyCurPos].classList.remove(freddy)
        freddyCurPos--
        cells[freddyCurPos].classList.add(freddy)
      } else {
        cells[freddyCurPos].classList.remove(freddy)
        freddyCurPos = freddyCurPos - width
        cells[freddyCurPos].classList.add(freddy)
      }
      collisionCheck(freddyCurPos)
    }
  } else {
    clearInterval(hereHeComes)
    cells[freddyCurPos].classList.remove(freddy)
  }
}
```

A further game feature I was determined to implement was the use of the player's character avatar at the bottom of the screen, with it changing expression to match the game events. This combined with the changing of the game's title to match the unusual game developments (and keeping in theme with the game's source theme of satire/absurdity) provides some extra details that might be missed by the player on a first playthrough, perhaps encouraging them to play again.

I have used functions that run on an interval to check for game conditions that then trigger progressive events in the game. For instance, the end of level 1 and 2 functions check for the presence of the frog on a particular grid index (index 9 i.e. top right corner for level 1 and index 94 for level 2), both of which are the starting indexes for levels 2 and 3 respectively, as well as being the respawn spots if the frog loses a life in levels 1 or 2.

There is also a collision function that is run after every enemy and frog movement to check for the presence of the frog and any other enemy within the same cells array index. The game over script is also included within this function as (for now at least) it is impossible for the player to lose in level 3 (it is just a little bonus round for now), and indeed the collision

function is effectively taken over by the hitCheck function, which checks for the presence of a laserbeam and an enemy in the same cell array index position.

Start screen
- Very simple title page with play button gif to start the game
- No instructions provided; I assumed the majority of people will be familiar with the frogger game mechanic and perhaps the game's theme

Intro screen/Prologue
- A short gif and audio file is played to encourage the player to show the game what you've got!

Level 1
- Theme music plays throughout level
- Objective to reach the portal appears fairly obvious but will review upon user feedback if any instruction might help

Level 1 end / Level 2 intro
- Short transition phase of the player travelling through the portal and indication that they are about to be chased!

Level 2
- Player starts where they left off but with slower enemies from previous level, with new enemy chasing them at a greater speed
- Objective is again implied, with a spinning suit of power armour in the place where they started level 1. Objective is to retrace your steps through the enemies while avoiding the chasing enemy (or not? ;o))

Level 2 end/Level 3 intro
- Short intro audio and gif to build up player for level 3

Level 3
- Player now powered up in their armour can wreak havoc on the frog army ahead of them and stop this nonsense before…
- Game controls are implied with use of two gifs of a hand hitting a space bar and a young man resembling our frog shooting a laser gun to provide instructions for what the player has to do

Game won/Epilogue
- Echoing the 'prologue', the character from the intro is pleased with the player
- Longer audio file and gif played in celebration of player completing the game

Game lost/Epilogue
- Game over screen implied by the character from the prologue not being impressed

Challenges
- enemyTwo movement
- Merging mechanics of three games into one
- Timings of transitions as audio files regularly are delayed in execution compared to visual changes. Adding image changes in the same timeout functions as audio changes does not mitigate this

Future improvement
- There are undoubted inefficiencies and repetitive code in this early stage. The structure has not been reviewed at all as once I introduced a new feature and got it working I swiftly moved on to the next task. I have, however, kept the game functions

in a fairly strict order, which became necessary after the game became over 500 lines(!)
- A restart button after losing all lives. After playing the games much during development, I figured out a strategy to never lose lives so rather neglected this part. After testing the game out on a few friends, it became clear that the game can be very difficult at first to reach level 2
- A third enemy type for levels 2 and 3, to be placed into the penultimate row on level 1. I had intended on having 3 different enemy distributions on the enemies, but decided to leave it at 2 as the last line of enemies on level 1 is quite challenging to pass. It takes some quick reactions to find a space to move quickly through. A simple revision of the enemy movement interval would allow a 3rd enemy to be present
- The chasing enemy in level 2 could be made to chase them incrementally quicker when the player gets further to the goal at the bottom
- Introduce some 'friendly floating log' rows like in the original Frogger, if this can be done without complicating the game's foundations (i.e. without the need to create an adjacent array or array within array, or use objects)
- Perhaps restructure the game by using classes and objects to handle 2 characters being on the same space at the same time and could possibly eradicate the laserbeam bug (more on that below)

Bugs

- enemyTwo has an invisible collision box at the end of the row above it. This is due to the way that the enemy movement is checked in a for loop ***after*** it has been moved but then immediately removed. This could potentially be mitigated against in the collision check function, but it would be better to address the enemy movement somehow
- The lasers that the frog fires in level 3 are bugged in that if the spacebar is pressed again before the current laserbeam onscreen 'despawns', the following laser blasts become 'super' and will destroy the entire column of enemies in one shot; also laserbeam litter can be left onscreen for the ending and until the end credits of the game

Copywright/Use of content not my own

I have used a lot of content in the game that is not mine and belongs to the original content owner. This game is a purely fan-made venture and just for fun/as a learning experience for the developer whilst learning on their software engineering course with General Assembly.