**Technical Design Document**

Frogger Game

*Course:* CSIS250 (Computer Graphics)

*By:* Ghina Bou Ghannam

*Date:* 09-05-2025

Table of Contents

[1. Introduction 3](#_Toc197723871)

[2. Architecture Overview 3](#_Toc197723872)

[3. System Requirements 3](#_Toc197723873)

[**1.** **Hardware Requirements** 3](#_Toc197723874)

[**2.** **Software Requirements** 3](#_Toc197723875)

[4. User Interface (UI) Design 3](#_Toc197723876)

[5. Modules and Components 4](#_Toc197723877)

[6. Algorithms and Data Structures 4](#_Toc197723878)

[7. Performance Optimization 5](#_Toc197723879)

[8. Error Handling and Logging 5](#_Toc197723880)

[9. Testing and Quality Assurance 5](#_Toc197723881)

[10. Documentation 5](#_Toc197723882)

# Introduction

This document outlines the technical specifications for Frogger game. The purpose of the project is to develop functional and engaging game that shows the understanding of game development principles including sprite management, collision detection, grid based movement, audio integration, and animation using HTML5 Canvas and JavaScript.

**Scope:** The initial scope of the project includes:

* Two Levels: The first level is an introduction to the game that is simple to figure out, while the second level adds new tasks to make it a little more difficult.
* Basic game mechanics: movement, collision detection, scoring, timer and grid based movement
* Audio: Implementation of sound effects and background music
* Instruction page
* Pause, Resume, Restart and Exit functionality
* Different types of obstacles and helpers: cars, turtles, logs…

# Architecture Overview

Frogger game follows component-based architecture where the components communicate together through the Game object which is the controller of whole gameplay. The main layers include *Game* Object which handles the update, draw, sprite, levels and input handling from keyboard and mouse. It also includes the *Sprite* object to handle drawing and updating the visual elements such as *Helper, Obstacle, Frog*... Moreover, *Level* Object to handle different initializations for each level. It also includes managing layers for *Score, Timer, Pause Menu, Win/Lose screens, How to Play screen, Main Menu*… The *Music Player* layer handles the background music and sound effects.

# System Requirements

## **Hardware Requirements**

The game runs on any mobile or desktop capable of running web browser

## **Software Requirements**

The game runs on different browsers such as Firefox, Safari and Chrome. The minimum screen resolution should be 650px width \* 700px height

# User Interface (UI) Design

**UI components:**

* Main game view is the canvas area of 650px width \* 700px height which contains all the game sprites.
* Score displays at the bottom right of the screen to show the gained points
* Lives display below the score at the bottom right of the screen as frog for each remaining life
* Timer displays at the bottom left of the screen to show the remaining seconds before time up
* Pause button displays at the top left of the screen to pause the game and show menu at the center of the screen including 3 buttons (Resume, Restart, Exit)
* Message appears for 1 sec at center of the screen when specific condition is triggered such as colliding with car, drowning in river…
* Road shows 5 lanes each of 50 px for cars moving in different direction at the lower part of the screen.
* Safe zone shows at the center of the screen for frog moving from road to river.
* River shows 5 lanes each of 50 px of logs and turtles moving in different direction at the top part of the screen
* Grass zone shows at the top of the screen above the river
* Goals show inside the grass zone at the top of screen

A single sprite sheet image was used containing all the graphical elements.

# Modules and Components

1. **Game**: engine that runs update/draw loop, manages sprites, and levels
2. **Sprite**: base class for all visual entities (frog, car, log…) with constructor, update and draw method
3. **Frog:** main player character that handles movement, collision with obstacles, interaction with helpers, updating the score and lives.
4. **Obstacle**: represents cars that move in different directions and speeds and hits the frog on contact
5. **Helper (Log/Turtle)**: helps the frog cross river where all logs float but some turtles drown periodically.
6. **Goal**: end zones where frog must reach to win the level
7. **Score**: tracks and displays the current score
8. **Timer**: countdown timer that loses the live if it ends before frog reaches goal and differs between each frog and different levels
9. **Music Player**: handles playing background music and sound effects
10. **Pause Button**: allows user to pause and restart or resume the game

# Algorithms and Data Structures

The algorithms and data structure used in the game for better functionality and performance includes:

* **Collision detection** determines if the frog overlaps with any obstacles or helpers. It is simple rectangle defined by x and y coordinates, so bounding box will be defined for each game entity. The two entities collide if their bounding boxes collide.
* **Movement logic** moves the obstacles and helpers linearly based on the speed and direction of each object. Then, for each update the x coordinate will update.
* **Grid based movement:** The frog moves in a fixed distance constrain. This movement ensures following the grid structure of the game world (river and road). The visual jump animation will occur between starting and ending grid cells.
* **Sprite management:** The sprites are added into array within the Game class to store all active game entities.

# Performance Optimization

To ensure a smooth and responsive gameplay experience, the game loop uses requestAnimationFrame() to synchronize rendering with the browser's refresh rate, leading to efficient animation updates. All necessary game assets, including images and audio files, are loaded at the start of the game to prevent any in-game loading struggles.

# Error Handling and Logging

The code will include logs for errors to the browser’s console, and try/catch blocks for potential errors such as audio.

# Testing and Quality Assurance

The testing strategies include:

**Unit Tests:**  testing individual components such as frog movement, collision detection, timer countdown, lives lose, and score increment

**Integration Tests:** testing the interaction between different elements such as the frog losing life when colliding with car, all the game stops when clicking pause.

The tool used for the Frogger game testing is manual testing via browser.

# Documentation

A clear and concise comments will be provided for functions and classes. Moreover, the folder structure will be clear such as images and audio will be under assets and each sprite is defined in different file.