VRIJE
UNIVERSITEIT
BRUSSEL

Project Computersystemen

Frogger

Group G23

Baris Sahan & Valentin Dominique Quevy

December 26, 2022

Academic year 2022-2023

Computerwetenschappen

1. Introduction

The purpose of this project was to create a video game using the assembly x86 language. We decided to recreate the classic game Frogger for this project. Frogger is a popular game in which the player controls a frog and must navigate through a series of obstacles to reach their goal. The project required us to have a solid understanding of the x86 assembly language and its capabilities. We also had to carefully plan and design the game mechanics and user interface. In the end, we were able to successfully recreate Frogger using x86 assembly language, providing a fun and challenging gaming experience for the user. This report will provide an overview of the development process for our Frogger game, including the design and implementation of the user guide, program features, and overall structure of the program. It will also discuss any challenges we encountered during the development process and how we addressed them.

2. How to play our game ?

To play our version of Frogger, the user must use the arrow keys on their keyboard to control the movement of the frog. The goal of the game is to guide the frog across a busy roadway and a flowing river, avoiding obstacles such as cars and using logs to reach the safe side of the map. The game is won when the player successfully guides four frogs across the map and into their designated homes at the top of the screen. The game will continue till you lose. Each frog that reaches its home will earn the player a point. The game is lost if a frog is hit by an obstacle or falls into the water. The player must navigate the frog carefully, as the obstacles on the roadway and river move at different speeds and in different directions. The game becomes more challenging as the player progresses and the obstacles move faster. Overall, the objective of the game is to use strategic planning and quick reflexes to guide the frog to safety and earn as many points as possible.

The escape key is used to quit the game, at any time.

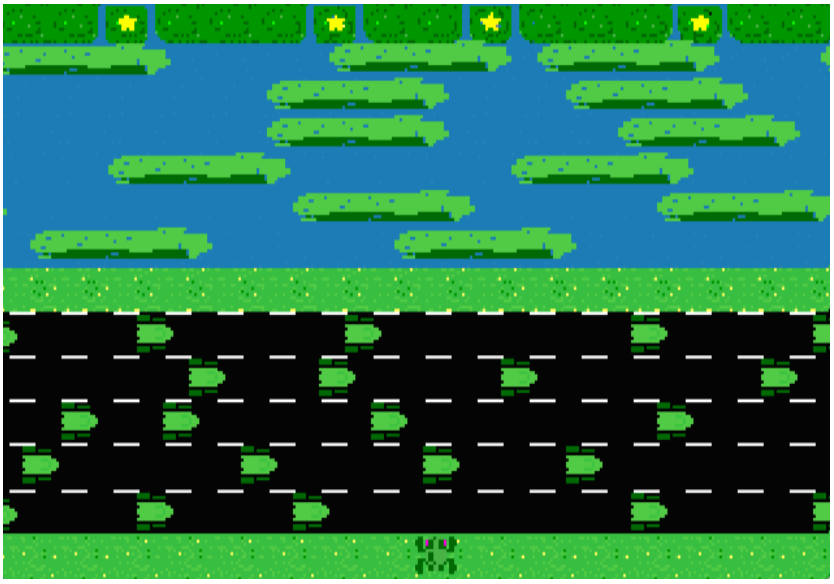

Figure 1: A screenshot of the game

3. Program features

The game is centered around a frog, which the player can move around the game using the arrow keys. There are also cars and logs present in the game that move separately from left to right or vice versa.

One of the key features of the game is the hit detection algorithm, which is used to detect collisions between the frog and the cars. If a collision is detected, the player loses the game. The hit detection algorithm is also used to detect when the frog is on a log, which allows the player to stay alive and continue playing.

To increase the challenge of the game, the program adjusts the speed of the cars and logs every time the player successfully places four frogs on the desired spots. This increases the player's score, as well as the speed at which the obstacles move.

To make the game more user-friendly, we have added a main menu screen that allows players to easily start the game.

Figure 2: Main menu screen

4. Program structure

4.1 Game Initialization

The main loop of our game begins by calling the init_game function, which initializes all the necessary elements for the game to run. This includes setting the video mode to 13h, updating the color palette to match the game menu, opening and reading the necessary files with the read_chunk function, and installing a keyboard handler to handle player input. Once these steps are completed, the game state is set to 0, which indicates that the game is in the menu state. The game state will be set to 1 when the ENTER-key is pressed, which indicates that the game is in the active gameplay state.

4.2 Game logic

Our main loop starts by calling the init_game function, which sets up all the necessary elements for the game to run. This includes setting the video mode, updating the color palette, opening and reading the necessary files, and installing a keyboard handler for player input. Once these steps are completed, the game state is set to 0, indicating that the player is currently in the menu. When the player presses the enter key, the game state changes to 1 and the game begins.

During the gameplay, the main loop constantly checks for keyboard input and passes it to the register for the program to react to. It then initiates the logic_game function, which handles the logic for both the menu and the game. In the menu part of the function, the screen is drawn and the function continues to loop until the player begins the game.

In the game part of the function, the program checks if any arrow keys have been pressed and adjusts the frog's position accordingly. It then calls other functions to handle the overall logic of the game, such as ensuring that the frog stays within the borders of the game and detecting collisions between the frog, cars, and logs. Before running the collision detection functions, the program first checks whether the frog is on the road or in the water, as this determines which collision detection functions to use.

4.3 Game video

The video_game function is an essential part of our game as it is responsible for updating the visuals that the player sees on their screen. The function is divided into two parts: one for the menu and one for the gameplay.

In the menu part of the function, the necessary graphics for the menu screen are drawn. This might include the title of the game, options for the player to select, and any other visual elements that are present on the menu screen.

In the gameplay part of the function, the program calls various functions to draw the elements of the game. This includes the frog, which the player controls, as well as the logs and cars, which are obstacles that the player must avoid or navigate around. The draw_moving_obstacles function is responsible for drawing and moving these obstacles as they travel across the screen. The program also draws the score, which increases as the player progresses through the game, and any frogs that are on the desired spots. All of these elements are drawn on the screen in real-time as the game is played, and the video_game function is called repeatedly by the main loop to ensure that the visuals are constantly updated.

5. Encountered problems

One problem we encountered was the flickering of the text on the game screen. This issue was caused by the way we were drawing our textual assets on the canvas. Every time the frog moved, we would redraw the entire screen, including the background, the obstacles, and the text. This process caused the text to flicker sometimes due to the frequent updates. To resolve this problem, we implemented a buffer in the form of a timer function. This function would wait a certain number of frames before performing another action, reducing the frequency of updates and eliminating the flickering. We were able to fix the issue by placing the timer function at different locations in the code and experimenting with different timer values to find the optimal number of frames to wait.
Another problem we had was how to detect our frog if it was on water or log. To solve this, we used the hit detection algorithm. We want to actually check if there is a collision between the frog and any other log. If there is a collision it means the frog is on log so the game should continue. If there is no collision between the frog and log, it means the frog is staying on top of the water and should lose.

6. Conclusion

We successfully developed a Frogger game using x86 assembly language, implementing various gameplay mechanics and features to provide a challenging and enjoyable experience for players. Despite facing some challenges during development, we were able to overcome them and are proud of the final product.