Group 01

CSC10003 OOP Project Documentation

Group members

Phan Hải Minh - 22127273
Bùi Tá Phát - 22127320
Đặng Thanh Tú - 22127432
Nguyễn Ngọc Anh Tú - 22127433

VIETNAM NATIONAL UNIVERSITY HO CHI MINH
UNIVERSITY OF SCIENCE
Faculty of Information Technology

Contents

I. Overview

This section overviews the gameplay, as well as other game features.

Introduction

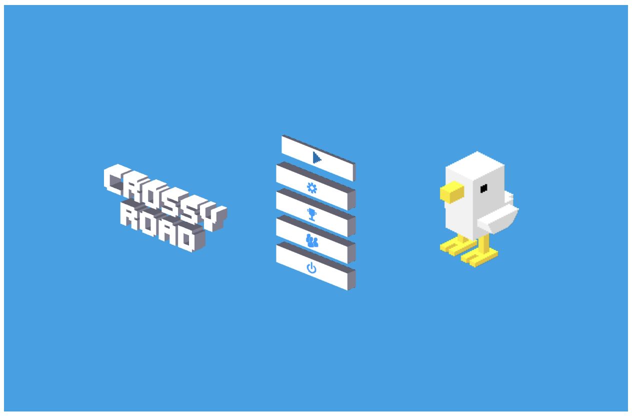
The game is about us, the character, who wants to cross roads and rivers to get to the grocery store but was intervened by the neverending traffic. We, the player, have to get our character safely to the other side, and we gain points as we progress through the roads. The player can use buttons from their keyboard to control the character, and the game is over once the character got hit by an obstacle. User can interact with the character or the elements on the screen using the arrow keys on the keyboard.

Gameplay

A game demo video can be watched here on Youtube.

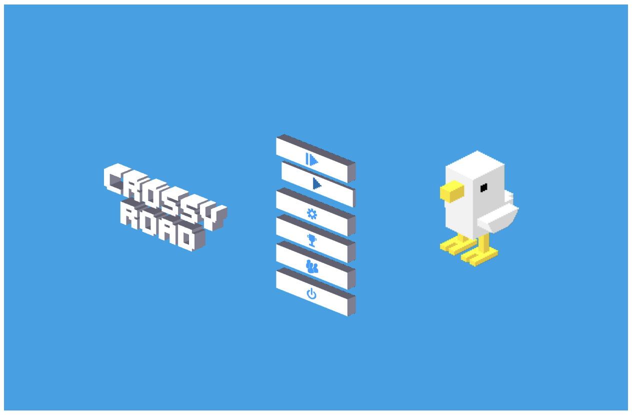
Menu

With the menu screen, the player can navigate through the menu using the up and down arrow keys and choose the button using the enter key. The menu screen has several buttons: Play, Setting, Leaderboard, Credit, and Exit.



Menu screen

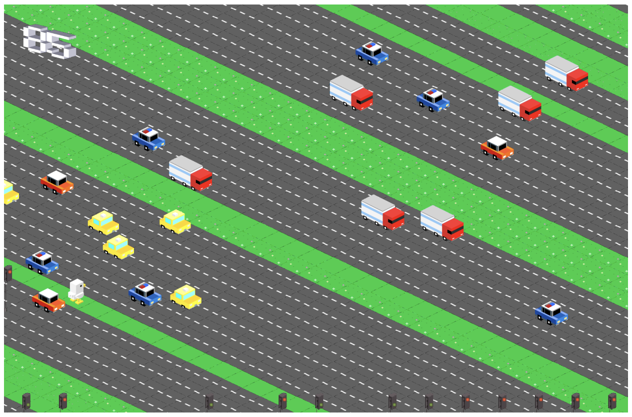
If the player paused the game from the previous session, a Continue button will appear on top of the Play button.



Menu with continue screen

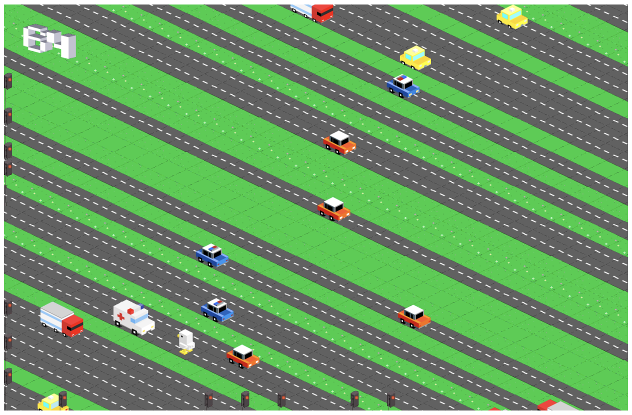
Play

The play screen is the main screen of the game. The player can use the arrow keys to move the character around the screen. The player can pause the game by pressing the escape key.



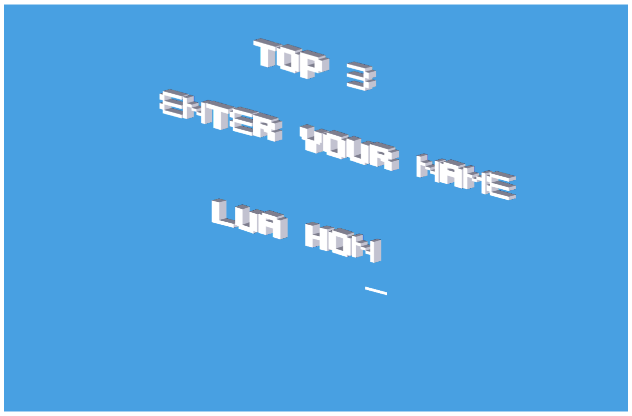
Gameplay

If the player hits an obstacle, an ambulance shows up and the game redirects the player to the game over screen.



Ambulance shows up

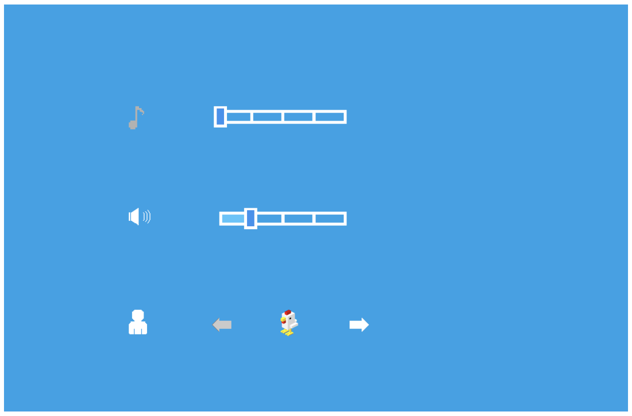
After a game, if the player makes it into the top 3 highest scores, the player will be asked to input a name to show on the leaderboard.



Name entering

Setting

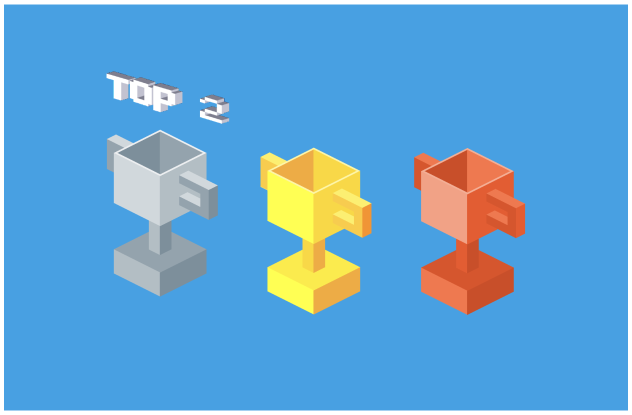
The setting screen allows you to edit the volume of background music, sound effects and lets you choose the sprite for your character. Like the menu, the user can iterate through the options using the up and down keys and change the value using the left and right keys.



Setting screen

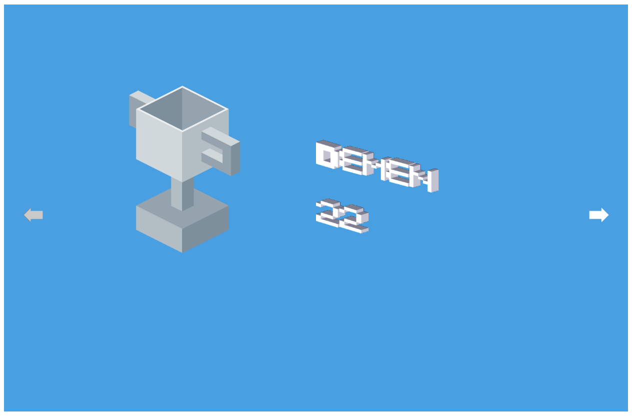
Leaderboard

The leaderboard screen has 3 cups: Gold, Silver, and Bronze. The player can use the left & right keys to iterate through the cups.



Leaderboard screen

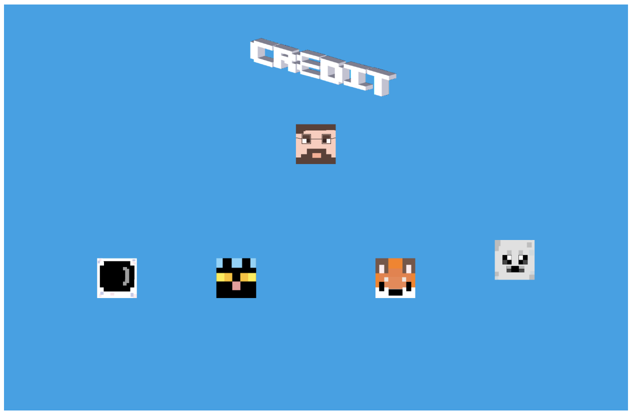
The player can choose a cup to view the name and score of that cup using the enter key.



Silver cup screen

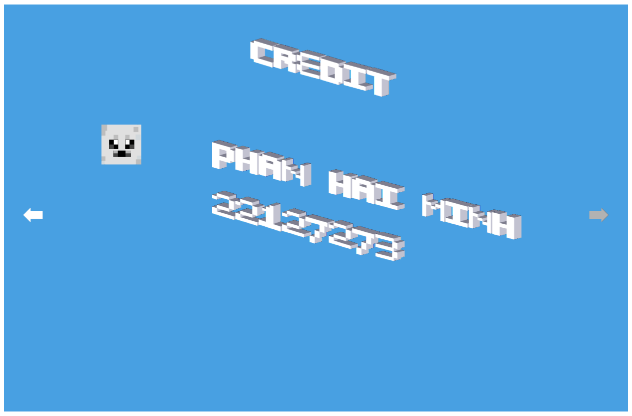
Credit

The credit screen shows the name of the group members and the lecturer.



Credit screen

Upon selecting an avatar, the full information of the member will be shown.



Credit information screen

Notable features

The game has several notable features:

- The game ultilizes the 3D-looking 2D textures, combined with linear algera to create a 3D-like projection. This is called isometric projection.
- The game saves the level when the player pauses the game. The player can continue the game even after restarting the game.

II. Technical details

This section describes the technical aspects of the game, as of how things work and how the game is implemented.

Code details

These are the classes that are used in the game.

Game internal structures

These are the classes that serves as the basis for internal game functionalities.

Name	Header	Description
`Color`	`color.hpp`	Contains an 32-bit integer for 4 color channels RGBA, serves as a pixel.
`Texture`	`texture.hpp`	An array of `Color` objects. Has width and height. Contains the texture of a game object.
`TextureHolder`	`texture_holder.hpp`	A class that holds all the textures of the game.
`Engine`	`engine.hpp`	A class that renders a texture to the screen.
`Vec2`	`lincal.hpp`	A class that represents a 2D vector. Used for transformations in isometric projecting.
`Mat2`	`lincal.hpp`	A class that represents a 2x2 matrix. Used for transformations in isometric projecting.
`Keyboard`	`keyboard.hpp`	A class that handles keyboard input.
`Sound`	`sound.hpp`	A class that handles sound data.
`Speaker`	`speaker.hpp`	A class that plays sound data in `Sound` objects.
`Setting`	`setting.hpp`	A class that holds the game settings. This class handles saving & loading settings from data files.
`Game`	`game.hpp`	A class that holds the game state. This class handles saving & loading game state from data files.

Game objects

These are the classes that represents objects that is involved in game logic.

Name	Header	Description
`Object`	`object.hpp`	A simple object, with a `Texture` and a pair of coordinates.
`Isometric`	`isometric.hpp`	Inherited from `Object`, this is an object that is rendered in isometric projection.
`Player`	`player.hpp`	A simple player, inherited from `Isometric`.
`Vehicle`	`vehicle.hpp`	A simple vehicle, inherited from `Isometric`. Also has the ability to move and check for collision with `Player`.
`Lane`	`lane.hpp`	A simple lane, contains a list of `Isometric` objects that act as blocks of the lane and a list of `Vehicle` objects.
`Traffic`	`traffic.hpp`	A simple traffic light, inherited from `Isometric`.
`Textbox`	`textbox.hpp`	A simple textbox, contains a list of `Isometric` objects that act as letters.

Game scenes

These are the classes that represents the scenes of the game.

Name	Header	Description
`Scene`	`scene.hpp`	A simple scene, has the ability to `process()` its objects and `render()` its objects.
`Menu`	`menu.hpp`	A simple menu scene, inherited from `Scene`.
`Play`	`play.hpp`	A simple play scene, inherited from `Scene`.
`Option`	`option.hpp`	A simple setting scene, inherited from `Scene`. This scene is named `Option` because `Setting` is already used for the game settings.
`Leaderboard`	`leaderboard.hpp`	A simple leaderboard scene, inherited from `Scene`.
`Credit`	`credit.hpp`	A simple credit scene, inherited from `Scene`.
`Gameover`	`gameover.hpp`	A simple gameover scene, inherited from `Scene`.
`SceneRegistry`	`scene_registry.hpp`	A class that holds all the scenes of the game.

Isometric projection

Linear algebra is used to project 2D textures onto the screen coordinates such that the textures overlap to create a pseudo-3D effect. The projection is done by applying a 2x2 transformation matrix to the in-game coordinates of the textures to obtain the screen coordinates (the top left coordinates which textures are rendered from).

The textures are all drawn so that the three coordinate axes appear equally foreshortened and the angle between any two of them is 120°. Since a "block" texture is a perfect cube, from the top left corner of the texture one can move half the texture width along the x axis and one fourth the texture height along the y axis to obtain the screen coordinates of the adjacent block texture. The transformation matrix is then as follows:

$$\begin{bmatrix} 0.5w & -0.5w \\ 0.25h & 0.25h \end{bmatrix}$$

where w and h are the width and height of the texture respectively. To obtain the screen coordinates, one multiplies this matrix with a 2D vector of the in-game coordinates of the object:

$$\begin{bmatrix} 0.5w & -0.5w \\ 0.25h & 0.25h \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

To shift the objects around on the screen, one can add a pair of offset coordinates to the screen coordinates to obtain the final screen coordinates. This is used for `Player` and `Vehicle` objects, which need to be drawn above the blocks instead of on the blocks to add height effect.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0.5w & -0.5w \\ 0.25h & 0.25h \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_{\text{offset}} \\ y_{\text{offset}} \end{bmatrix}$$

Game looping logic

Each different scene of the game is represented by a `Scene` class. The `Game` object holds a pointer to the current scene and a `SceneRegistry` object that holds all the scenes of the game.

Each `Scene` object contains all the game objects that are involved in the scene. The `render()` function renders all the objects of the scene to the screen, and the `process()` function does the logic processing between the objects in that scene. The `process()` function returns a pointer to the next scene that the game should render. If that scene is to be rendered again, the `process()` function returns a `this` pointer to itself.

In each loop, the game renders the current scene using the `render()` function of the scene. The game then calls to the `process()` function to receive the next scene it should render. The game ends when the `process()` function returns a `nullptr` pointer.

Game saving

The game saves the game state in a file called `gamestate.dat` and the game settings in a file called `setting.dat`. The game state is saved in binary format, while the game settings is saved in text format.

Format of `gamestate.dat`

Special padding: 3 bytes

These data is used to check if the file contains gamestate or not

```
0000 0000  0000 0000  0000 0000
---- ----  ---- ----  ---- ----
D    E     F    C     A    D
```

Address of read pointer a: 32 bytes (= 8 address at most, each address is an integer)

- These data will be stored right after the checking padding, it use for seek the read pointer for easier read data because there are some data that doesn't have a specifically size like name of player (type string) and more.
- Address of score and name: 4 bytes
- Address of lanes' data: 4 bytes
- Address of player's data: 4 bytes
- Address of other objects' data: 4 bytes
- Padding: 16 bytes

Score and offset: 8 **bytes**

- Score: 4 bytes (`gamestate[0][0]` to `gamestate[0][3]`)
- Offset: 4 bytes (`gamestate[0][4]` to `gamestate[0][7]`)

Lanes: $17N$ **bytes**

- Number of lanes: N (this isn't a stored data)
- Lanes data: $17N$ bytes (`gamestate[1][0]` to `gamestate[1][17 * N - 1]`)
  - y coordinate: 4 bytes
  - Speed: 4 bytes
  - Spawn clock: 4 bytes
  - Traffic state: 1 bytes
  - Traffic clock: 4 bytes

Player: 5 **bytes**

- The numerical order of lane: 1 bytes (`gamestate[2][0]`) (`lanes[this]` has this player)
- x coordinate: 4 bytes (`gamestate[2][1]` to `gamestate[2][8]`)

Cars, trucks: $5N$ **bytes**

- Number of cars (and trucks): N (this isn't a stored data)
- Objects data: $9N$ bytes (`gamestate[n][0]` to `gamestate[n][9 * N - 1]`)
  - The numerical order of lane: 1 bytes (`lanes[this]` contains this vehicles)
  - x coordinate: 4 bytes

Format of `setting.dat`

Volumes & sprites

These data will be stored at the start of the file:

- A 3-byte magic padding number is used at the beginning for a quick check of the file's validity. The value `0xDECADE` is used because it remains the same no matter the endianness of the system (reading from the most significant byte or the least significant byte is the same), and it also spells out "decade" which is nice. The remaining byte is used to contain 2 `Volume` types and a `Sprite`.

- A `Volume` type has 5 possible values (`min`, `low`, `medium`, `high`, `max`) which takes up 3 bits, hence 2 `Volume` types (one for music, one for SFX) can be stored with 6 bits. Since the `Sprite` enum current only has 3 options of `duck`, `chicken` or `cat` (3 possible values), it can be stored with the remaining 2 bits, filling a full byte.

Follow up these data are the highscores. The format of this section is as follows:

```
0000 0000  0000 0000  0000 0000  000 000 00
---- ----  ---- ----  ---- ----  --- --- --
D    E     C    A     D    E     MUS SFX SP
```

Highscores

The game has 3 highscores for 1st, 2nd and 3rd place, each highscore is a `word` (`unsigned int`) type. Storing these highscores are straightforward as follows:

```
0000 0000 0000 0000 0000 0000 0000 0000
---- ---- ---- ---- ---- ---- ---- ----
1st place (4 bytes)

0000 0000 0000 0000 0000 0000 0000 0000
---- ---- ---- ---- ---- ---- ---- ----
2nd place (4 bytes)

0000 0000 0000 0000 0000 0000 0000 0000
---- ---- ---- ---- ---- ---- ---- ----
3rd place (4 bytes)
```

Since the game only allows the player to enter at most 8 characters for their name, each name only takes up 8 bytes. The names are then saved consecutively after the highscores.

III. References

Link to this project repository

- https://github.com/hydroshiba/crossy-clone

Isometric projection

- https://en.wikipedia.org/wiki/Isometric_projection
- https://www.youtube.com/watch?v=04oQ2jOUjkU

Art drawing

- Texture drawing website: https://www.pixilart.com/draw
- Converting PNG to 32-bit BMP: https://online-converting.com/image/convert2bmp/

Sound effects

- https://freesound.org