Slovak Technical University in Bratislava

Faculty of Informatics and Information Technology

Fundamentals of creating multimedia applications

The Last Hope

Control Point I

Lucas Daniel Espitia Corredor

Time practice:: Streda 16:00-18:00

Ing. V Kunštár

Date: 05/03/2022

1. Description of the basic objectives and operation of the game

This assignment is for a university project, specifically for the summer semester. The student is expected to apply the knowledge acquired in the course to implement a fully functional game in JavaScript, using object-oriented programming techniques and the tools learned in the classroom.

The project will be evaluated based on its functionality, complexity, originality, and the student's ability to explain and justify the design decisions made during the development of the game. The student is expected to deliver the project on time and actively participate in the presentation of the project during each checkpoint established during the course.

1.1 Concept

The game to be developed is a customized version of the classic arcade game Pac-Man. In this version, the player must navigate through a maze to collect points, while being chased by zombies. Unlike the original game, in this case, the player will have the ability to use weapons to attack enemies.

1..2 Objectives:

The game's main objective is to provide a fun and exciting experience for the player while learning basic JavaScript programming concepts and practicing game development skills. The main objective of the game is to complete all levels of the maze without losing lives.

1.3 Requirements:

To develop this game, knowledge in object-oriented programming (OOP) in JavaScript is required. Knowledge in HTML, CSS and problem solving skills are also required, as well as the ability to work autonomously.

1.4 Specifications:

The game will feature multiple levels, each with a unique and challenging maze design. The player will control the main character, Pac-Man, and must collect all the dots in each level while avoiding zombies. In addition, the player will have the ability to collect weapons and use them to attack the zombies.

The game will also have a scoring system, where points will be awarded for collecting points, for eliminating zombies and for completing each level. There will be a lives system, where the player will start with a limited number of lives and lose one life each time they are caught by a zombie.

The game will have an intuitive and easy-to-use user interface, with a main screen displaying game start options, sound settings and other option menus. There will also be screens to display the results of each level and the high scores achieved by players.

The game will feature a system where the computer will control the zombie monsters, giving the player an additional challenge of dodging enemies while collecting points and weapons. The zombies will move autonomously through the maze and will chase the player once they have detected them.

The game will be divided into three different levels, each of which will be more difficult than the last. Each level will have a unique and challenging maze design, which will force the player to think strategically about how to avoid the zombies and how to successfully complete the level.

The game will feature a system where the computer will control the zombie monsters, giving the player an additional challenge of dodging enemies while collecting points and weapons. The zombies will move autonomously through the maze and will chase the player once they have detected them.

2. Game control design

The control design of the game is based on the user's interaction with the main character, Pac-Man. The player will control Pac-Man by using the arrow keys to move through the maze and avoid the zombies. In addition, the player will be able to pick up weapons and use them to attack the zombies, the weapons are fired with the space key and pointed in the direction Pac-Man is moving.

The control design also includes a pause system, which will allow the player to temporarily pause the game at any time. The pause system will be activated by pressing the "ESC" key and will display a menu of options that will allow the player to resume the game, exit to the main menu or adjust game settings.

The game will also include an option to adjust the sound settings, which will allow the player to adjust the volume of the game's music and sound effects. This option will be displayed in the pause menu and will also be accessible from the main menu of the game.

Also for the project, it has been decided to implement in the menu part, a kind of "cheat" in which you can see the different levels, this in order to be able to appreciate each level, of course the main idea was to start from the easiest and with each level gets harder, this is done to be more practical.

In summary, the control design of the game focuses on providing the player with an intuitive and easy to use gaming experience. The arrow key system for moving the main character, along with the ability to pick up weapons and use them to attack zombies, gives the player complete control over the game. The pause and sound adjustment option also enhances the user experience by allowing them to customise the game to their liking.

3. Design of all game screens and currency

The design of all screens and menus in the game is an essential part of the development process, as it is what connects the player to the game experience itself.

The usability, clarity, and aesthetics of the game have been considered in the design of each screen.

3.1 Main menu

The main menu is the first screen the player sees when starting the game. On this screen the player is presented with the options to start a new game, continue a previous game if it has been saved, access the game settings, view the levels available to the player and something about me.

3.2 Level selection
In this screen, the player will have the possibility to select the level he/she wishes to play. Levels will become available as the player progresses through the story.

3.3 Game screen
The game screen will show the main action of the game. Here, the player will control his character in the game world and face different challenges.

3.4 Pause screen
The pause screen will be activated when the player presses the ESC key. In this screen, the player will have the possibility to continue playing, return to the main menu, adjust the volume of the music or sound effects, or exit the game.

3.5 Settings
In the settings screen, the player can adjust the volume of the music and sound effects, go back to main menu, reset the game, continue the game.

3.6 Cinematics
A cinematic screen will be added to explain the story behind the apocalypse the player character is in, for this part we will include slides in which the mini-story of the game will be told, in order to give the player context and a greater purpose.

3.7 Game Over
Designed to effectively show the player that they have lost. A darker, scarier design style will be used to convey the grim atmosphere of the game. The Game Over screen will also offer the player the option to restart the level or return to the main menu.

Each screen and menu has been designed with a consistent and appealing graphical style for the player. Dark colors have been used to create an atmosphere of tension and mystery throughout the game.

4. Basic OOP description of objects in the game (classes, their attributes, and methods)

4.1 Survivor
This class represents the player-controlled character. It has the following attributes:

- **lives** is an integer indicating the number of lives remaining.
- **speed:** an integer indicating the speed of movement.
- **direction:** a string indicating the current direction of the character.
- **position:** a tuple indicating the character's current position on the map.
- **weapon:** an instance of the Weapon class indicating the current weapon the character has equipped.
- **score:** an integer indicating the player's current score.

Methods:

- **move():** moves the character in the indicated direction.
- **fire():** causes the character to fire its current weapon.
- **change_weapon():** changes the character's current weapon to the one specified.
- **lose_life():** decreases the number of lives of the character by 1.
- **gain_points():** increases the player's score by the given amount of points.
- **die():** displays the Game Over screen when the player loses all lives.

4.2 Zombie

This class represents the enemies of the game. It has the following attributes:

- **Life**: Indicates the amount of life remaining from the zombie.
- **Velocity**: Indicate the speed of movement.
- **Position**: Indicate the current position of the zombie on the map.
- **Address**: Indicate the current address of the zombie.

Methods:

- **Move()**: Moves the zombie in the indicated direction.
- **get_damage()**: decreases the life of the zombie by the amount of damage indicated.
- **Die()**: Removes the zombie from the map when his life reaches zero.

4.3 ZombieBoss

This class represents the mini-boss that appears after reaching a certain score. It has the following attributes:

- **Life**: An integer that indicates the amount of life remaining from the boss.
- **Velocity**: Indicate the speed of movement.
- **Position**: Indicate the boss's current position on the map.
- **Address**: Indicate the current address of the boss.

Methods:

- **Move()**: Moves the boss in the indicated direction.
- **get_damage()**: decreases the life of the boss by the amount of damage indicated.
- **Die()**: Eliminate the boss from the map when his life reaches zero.

4.4 Weapon

This class represents the different weapons that the player can use. It has the following Attributes:

- **Type**: Indicate the type of weapon.
- **bullet_quantity**: Indicate the number of bullets remaining.
- **Time_use**: Indicate the amount of time the weapon can be used before disappearing.
- **Position**: indicate the current position of the weapon on the map.

Methods:

- **Shoot():** This method allows the player to fire a projectile from the equipped weapon in the direction in which it is moving.
- **Reload():** This method reloads the ammunition of the current weapon. It can only be reloaded if the weapon's ammunition is exhausted and if the player has spare ammo.
- **Get ammo ():** This method returns the current amount of ammunition the weapon has.
- **getTimeRemaining():** This method returns the amount of time remaining before the current weapon is depawned.

4.5 Cure

Attributes:

- **pos_x**: X coordinate in which the cure is located
- **pos_y**: Y coordinate in which the cure is found
- **Value**: value of the cure, indicating how many healing points it awards

Methods:

- **init()**: constructor that initializes the attributes of the cure
- **get_pos()**: returns a tuple with the X and Y coordinates of the cure
- **get_valor()** : returns the cure value

4.6 Wall

Attributes:

- **pos_x**: wall X coordinate
- **pos_y**: wall Y coordinate

Methods:

- **init()**: constructor that initializes wall coordinates
- **get_pos():** Returns a tuple with the X and Y coordinates of the wall

4.7 Score

Attributes:

- **Value**: Current value of the score

Methods:

- **init()**: constructor that initializes the score value to 0.
- **grow_score()**: increases the score value by the amount indicated by value.
- **get_score()**: returns the current value of the score.

4.8 Lives

Attributes:

- **Quantity**: Current number of lives

Methods:

- **init()**: constructor that initializes the number of lives at the value.
- **rest_life()**: subtracts one life from the current number of lives.
- **add_life()**: adds one life to the current number of lives.
- **get_life()** : Returns the current number of lives.

5. GRAPHICS AND SOUNDS:

5.1 GRAPHICS:

For the creation of the game, several graphic elements have been chosen according to the needs that are required to create a terrifying environment, an atmosphere of tension, among other elements that are part of the game, so the following textures have been implemented:

5.1.1 Logos:

First of all, we have the official logos created by me, they have a great meaning since they provide "my brand" or my nickname, as well as the logo of the videogame.

1. Nickname "DanTV" Logo

2. Videogame Logo

1. *Oficial logo from my "brand".*
2. *Oficial logo for the Videogame*

5.1.2 Vector Images:

These images mostly correspond to the interactive menus, the sound and certain components within the game that have not required much time to create, some of these images have been created by me, others have been downloaded from the internet but care has been taken and none of them have copyright problems.

3. Settings/ Pause

4. Mute Volume

5. Volume on

3. *Menu, to pause, interact in the game, to change settings, volume, etc.*
4. *Volume Off*
5. *Volume On*

6. Ammo

7. Health

8. Bullet

6. Ammo, the player must pick em up to use his weapon, reappears in a time interval.

7. The life counter

8. Projectile of the gun, it will move according to the fired direction.

5.1.3 Survivor Image:

For the main character a 2D shape design has been created including walking animations, depending on the direction he is going, only included: up, down, right, left.

9. Survivor 1

10. Survivor 2

11. Survivor 3

12. Survivor 4

In my ideas, I plan to make each character have certain characteristics unique to others, for example recharge time, movement speed, etc. This idea is under consideration and waiting

if the teacher allows me and if he advises me, since the idea is not to make a complicated videogame.

I would like to include a section where the player can choose between several characters, this section is under consideration and if possible these will be the options that the player can have for his survivor.

5.1.4 Enemy Image:

In the same way as the survivor, the zombies have been created in 2D form, adding up, down, right and left walking animations.

13. Zombie 1

14. Zombie 2

15. Zombie 3

16. Zombie 4

17. Zombie boss

In total there are 5 zombies, 4 that are the main enemies and another one that will be the final boss.

At this point it is necessary to remember that the previous images were downloaded from OpenGameArt.org although it is not necessary to make a contribution to the author, here I leave his tag, thanks to him I have been able to have the styles of my characters and my enemies.

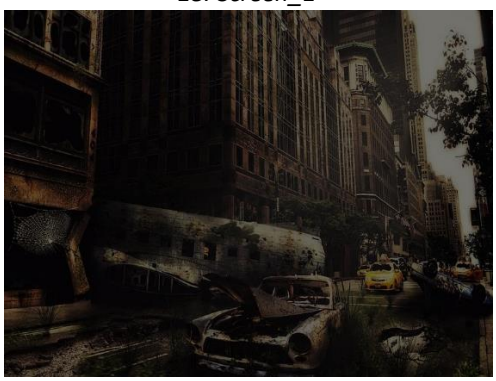
Contribution to: Curt, cjc83486. https://opengameart.org/content/zombie-rpg-sprites

5.1.5 Screen Pictures

Images have also been introduced as wallpapers to give the game a creepier and more frightening atmosphere, and have been carefully chosen in a way where they have no copyright and it is not necessary to make any contribution.

18. Screen_1

19. Screen_2



20. Screen_3

21. Screen_4

22. Screen_5

23.Screen_6

24. Screen_7

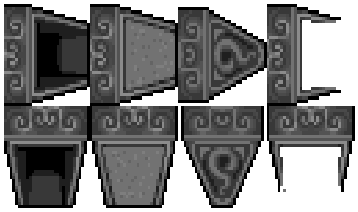
25.Gameover_Screen

26. General Background

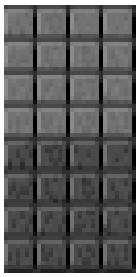
All images were carefully chosen to have the look and feel of horror, and will be implemented with CSS and JavaScript properties so that they can be loaded in a loop. The Gameover screen and the general background screen are the only ones that will not be in a loop.

5.1.6 EXTRAS:

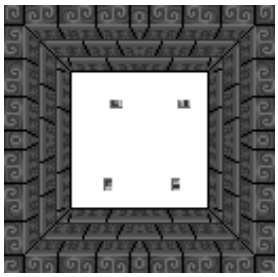
Images that have importance and are part of the graphics to be used.



27. Doors



28. Floor



29. Walls

27. Used at the exit of the zombies and on the sides where pacman returns to the maze.

28. Where the player and the zombies can walk on.

29. Places where the player and the zombie cannot pass through

30. Blood

31. Grave

30. Effect produced when a bullet is merged with a zombie

31. Image that appears when you kill a zombie, when you pass through it you get the points for the death of the zombie.

5.2 SOUNDS

All the music and sound effects have been carefully chosen, taking into account that the game must convey the atmosphere of terror, as well as all this part is free of copyright.

Although it is not possible to insert the sounds in this type of file, I am going to attach a Zip file, where you can find my sounds, however, here I leave a brief description of the chosen sounds, both musical and effects.

5.2.1 Music

The following music has been chosen in order to immerse the player in a terrifying atmosphere, the following tracks are based on transmitting tension, terror, hopelessness, etc.

This music will be played constantly, except in certain occasions, for example: The Gameover screen and the cinematics, also this music can be muted from the main menu or the pause button within the game.

GENERAL MUSIC

I. Cinematic-dramatic-11120.mp3:

cinematic-dramatic-1
1120.mp3

II. Zander Noriega–Article196Section1.mp3

Zander Noriega -
Article 196 Section 1.v

III. Alexander Ehlers - Flags

Alexander Ehlers -
Flags.mp3

IV. Alexander Ehlers - Great mission

Alexander Ehlers -
Great mission.mp3

V. Alexander Ehlers - Waking the devil

Alexander Ehlers -
Waking the devil.mp3

VI. мистичная тема

мистичная тема.mp3

CINEMATICS

I. dark-mystery-trailer-taking-our-time-131566_final_level

dark-mystery-trailer-taking-our-time-13156

II. exiting-strings-21668

exiting-strings-21668.mp3

GAMEOVER

I. Gameover

gameover.mp3

II. ambient_horror_track01

ambient_horror_track01.wav

WINNING

I. Medicine

Medicine.mp3

5.2.2 Sounds Effect

I. Click

buttons.mp3

It is activated when the user interacts with the buttons of the main menu or the menu that pauses the game.

II. *Footstep*

footstep.ogg

It is activated when the user moves up, left, down, right. Each movement is one step.

III. *Shoot*

9mm-pistol-shot-6349.mp3

When the user shoots his/her gun

IV. *Reload*

reload.wav

When the player reload his/her weapon.

V. *Get Life*

When the player gets an amount of points, he/she will get a life, that is when the sound gets activated.

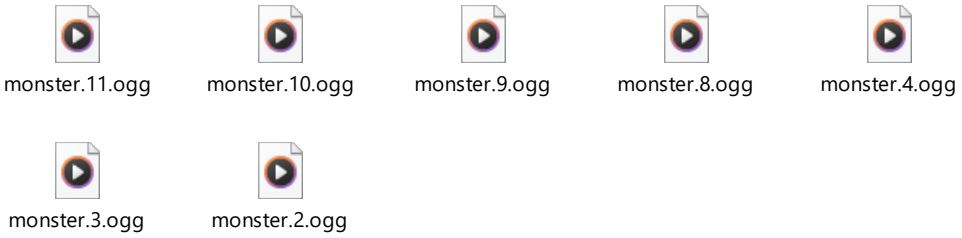
life_get.mp3

VI. Bullet Collision

collision.wav

When the bullet gets an enemy or the Wall

VII. **Zombie moaning.**



Different zombie sounds that will be played every certain time interval.

6. CONTROL POINT 4

6.1 CLASSES

6.1.1 Survivor

The constructor of the "Survivor" class defines the properties of the "Survivor" object. These properties include the character's position on the X and Y axis, its size, speed, direction, animations and other important variables. The image cache is also loaded in the "loadImages" method and an interval for changing the character's animation is set in the "changeAnimation" method.

The "moveProcess" method is responsible for changing the direction of the character and moving it forward in the corresponding direction. If there are collisions, the character will move backward in its current direction.

The "getPoints" method checks if the character has eaten something (a piece of food or diamond) and if so, updates the score.

The "moveBackwards" and "moveForwards" methods are responsible for moving the character backward or forward in its current direction, respectively.

The "checkCollisions" method checks for collisions in the character's current direction. If there are collisions, it returns true; otherwise, it returns false.

The "checkBounds" method is responsible for checking if the character has left the screen boundary. If so, it adjusts the character's position so that it reappears on the screen.

The "checkZombieCollision" method checks if the character has collided with a zombie. If so, it returns true; otherwise, it returns false.

The "changeDirectionIfPossible" method changes the direction of the character if possible. It first checks if the current direction and the desired direction are the same. If they are the same, it does nothing. If they are different, it tries to move the character forward in a new direction. If there are no collisions, it updates the current direction of the character. If there are collisions, it returns to the previous direction.

The "changeAnimation" method changes the animation of the character by calling the next image in the image cache.

The "changeAnimationFrames" method changes the animations of the character according to the direction in which it is moving.

Finally, the "draw" method is responsible for drawing the character on the canvas. First, the state of the current canvas is saved. Then, the position of the character is adjusted so that it is in the center of the current block. Finally, the current image is drawn on the canvas and the previous state of the canvas is restored.

6.1.2 ZOMBIES

`Zombies` class represents a zombie character. The class takes several arguments, including the zombie's position, size, speed, animation frames, and range. The class has several methods, including `moveProcess()`, `moveBackwards()`, `moveForwards()`, `checkCollisions()`, `checkBounds()`, `isInRange()`, `changeDirectionIfPossible()`, `calculateNewDirection()`, and `addNeighbors()`.

The `moveProcess()` method is used to move the zombie character. The method first checks if the zombie is within range of the survivor character. If the zombie is within range, the survivor becomes the zombie's target. If not, the zombie picks a random target from an array of targets. The zombie then attempts to move toward the target by changing its direction using the `changeDirectionIfPossible()` method and moving forwards using the `moveForwards()` method. If the zombie collides with an obstacle or wall, it moves backward using the `moveBackwards()` method.

The `checkCollisions()` method checks if the zombie has collided with an obstacle or wall. The method checks the coordinates of the four corners of the zombie's bounding box against the map of the game and returns `true` if any of the corners collide with an obstacle or wall.

The `isInRange()` method checks if the zombie is within range of the survivor character by calculating the distance between the two characters and checking if it is less than or equal to the zombie's range.

The `changeDirectionIfPossible()` method changes the direction of the zombie towards the target if possible. The method first calculates the new direction using the `calculateNewDirection()` method and then attempts to move the zombie in that direction. If the zombie collides with an obstacle or wall, the method moves the zombie backward and reverts the direction change.

The `calculateNewDirection()` method calculates the new direction for the zombie using a breadth-first search. The method takes the game map and the coordinates of the target as arguments and returns the new direction.

The `addNeighbors()` method adds the neighbouring cells of the current cell to the queue for breadth-first search. The method takes the current cell and the game map as arguments and returns an array of neighbouring cells.

6.2 MVC

6.2.1 MODEL

There are 4 javascript files in the model

Constants.js

In this one are stored all the variables of the game, all the identifiers obtained by the use of ID, booleans that are needed for the realization of processes, the variables of the game, the different variables for each level, as well as the music and the images related to the frames.

- ***Preload.js***

This script has everything related to images and their preloading, this in order to avoid loading bugs.

- ***Survivor.js***

This script contains all the information of the Survivor class, described above.

- ***zombies.js***

This script contains all the information of the zombie class described above.

6.2.2 VIEW

This directory is divided into two other subdirectories, screen and video game...

SCREEN

- ***cinematics.js, gameover.js, levels.js,***

Contains everything related to the cinematics view, divs, buttons, main HTML, and some functions that change the layout and are used in the controller.

- ***credits.js, settings.js, tutorial.js, win.js***

Contains all the HTML code for the credits div.

- ***music.js, screenStart.js***

Contains only functions, in order to make a change in the screen, are called from the controller.

VIDEOGAME

- ***map.js***

It contains everything related to the creation of figures in the canvas for the creation of the map, as well as a sketch with numbers and its function to create the map...

The number 1 means wall, 2 is the path where the survivor passes and there the points or food are created.

- ***menu.js***

In this JavaScript file, the complementary menu is drawn, that is to say, the menu that appears in the lower part of the videogame.

- ***points.js***

This file handles the creation of the food, the reset, and the operation of the corresponding sprite.

CONTROLLER

The controller is also divided in two, one for the video game and the other for the screen.

Screen

- ***cinematics.js***

This javascript is in charge of recreating the cinematic of the beginning, it works mainly by a click event.

- ***credits.js, gameover.js, settings.js***

It is in charge of the part of the credits, it gives the corresponding html, introduces the code, and has functions inside or events that allow to make more things in this...

- ***level.js, tutorial.js***

Same as the previous one but it also has some sprites, the reason why it is not made in the view is that the ids of change are obtained only inside the function.

- ***music.js***

It is in charge of all the reproduction, mute, pause, etc that is related to the music and the effects of the video game.

VIDEOEGAME

- ***back_menu.js***

Handles the event of returning to the menu once it is confirmed.

- ***controls.js***

Contains the key events, everything related to the game and some specific cases.

- ***elements.js***

This file is in charge of creating all the elements, and to obtain some logic for the main cycle, in this one is the creation of the Survivor, and its update, the creation of the Zombies and some other functions that are necessary in the game.

- ***gameover.js***

It is responsible for resetting the game, every time the game is lost, or contains all functions related to losing.

- ***level.js***

Contains the logic to be able to change levels, as well as the interface between levels.

- ***win.js***

It is in charge of showing the WIN condition.

- ***main.js***

Contains the main cycle of the game, its maintenance, update of the functions and the drawing.

PUBLIC (EXTRAS)

Contains the extra files for the sounds, music, images, etc… It also contains the CSS and one javascript file “background.js”, this one just makes the carrousel for the background.

Comparison

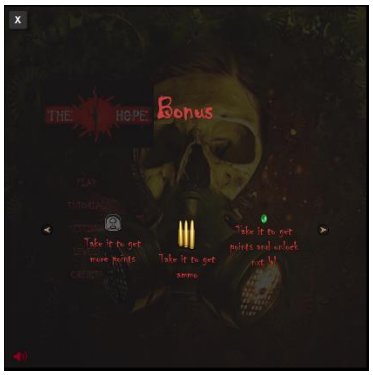
Although the general idea was broader, on the way through this project I ran into more bottlenecks than I thought, certain methods I wanted to introduce or ways to make the game differently could not become a reality.

Despite that, some elements did become reality as I thought, unfortunately, I never added frameworks at the beginning of the project so it can't be compared, unfortunately.

Pictures from the game

BEGINING

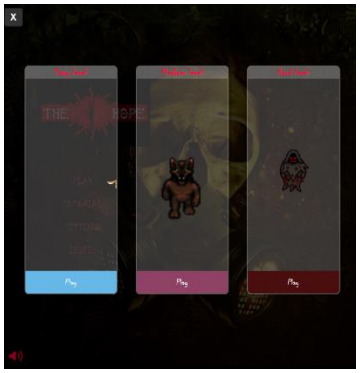
TUTORIAL



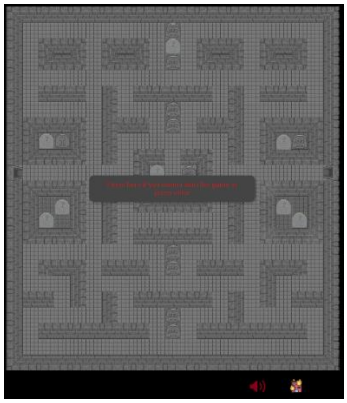
Settings

Credits

Levels



Preload game



Game

Music Panel

Gameover

Menu in game