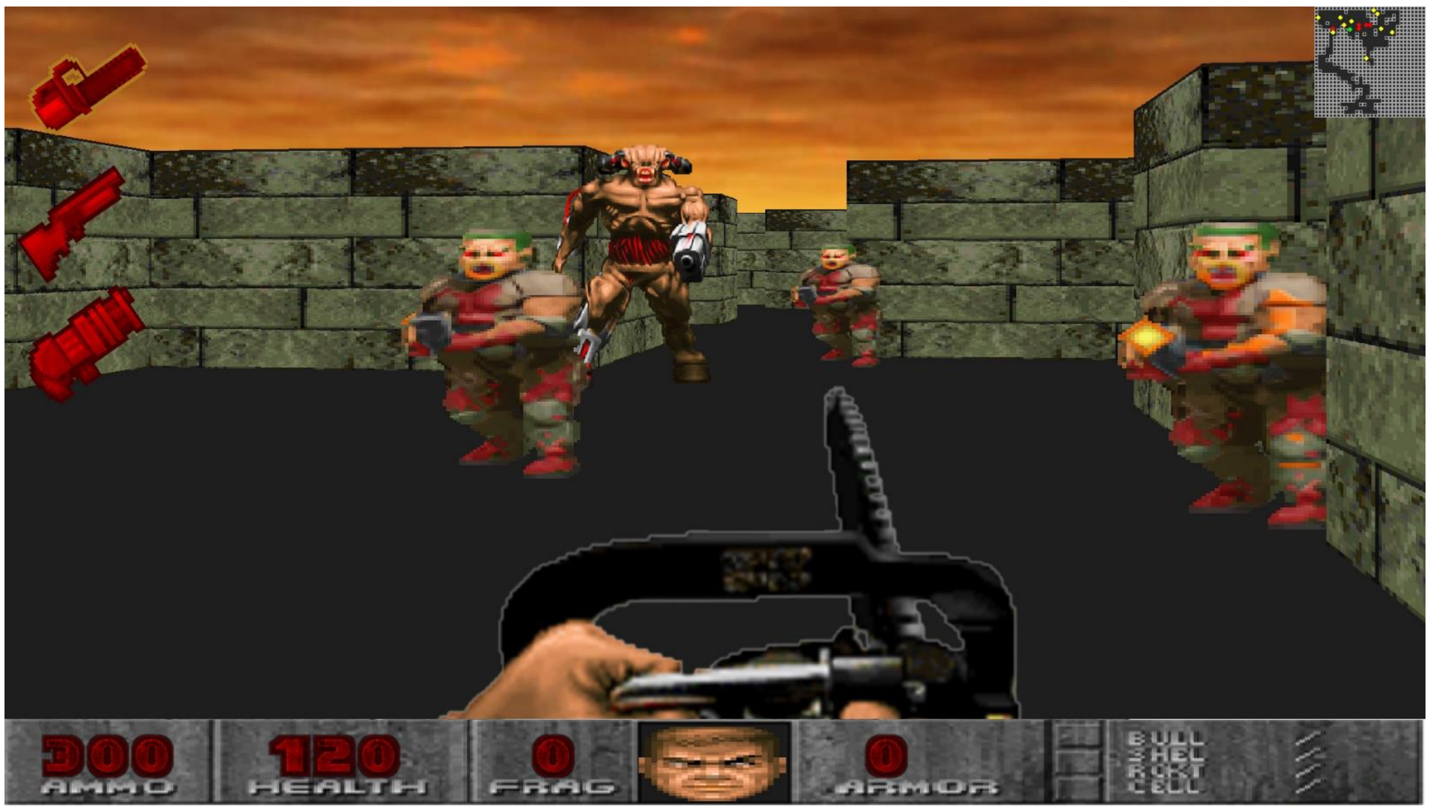
Table of contents

3. Code overview

- **Wolfenstein Copy**
    - button module
    - hud module
    - interactions module
    - main module
    - main_menu module
    - map module
    - npc module
    - object_handler module
    - path_find_bfs module
    - player module
    - random_map_generation module
    - raycasting module
    - renderer module
    - save_and_load_game module
    - settings module
    - sound module
    - sprite_object module
    - weapons module

Brief overview

Shorts Description



The game is an FPS shooter Wolfenstein and Doom like clone. The main purpose of this project is to lay foundation for further implementation and innovation of the used technologies. Of course, it can be used as a base to make even more detailed and finished games. The main goal of the game is to kill as many enemies as possible and to survive for longer period of times.

Technologies used

1. Python – 3.9.4
2. Pygame – 2.1.2
3. Numpy – 1.23.1
4. Pathfinding – 1.01

Requirements

Operating system: Windows 10 2022 64 bit or Linux Mint 21, Ubuntu 22.04

Hardware: Monitor, mouse

Dependencies: Python 3.9.X, Pygame 2.1.X, Numpy 1.23.X

Installation

You can clone the respiratory from here, or you can download the executable from here.

If you choose to clone the respiratory. You can do in your terminal:

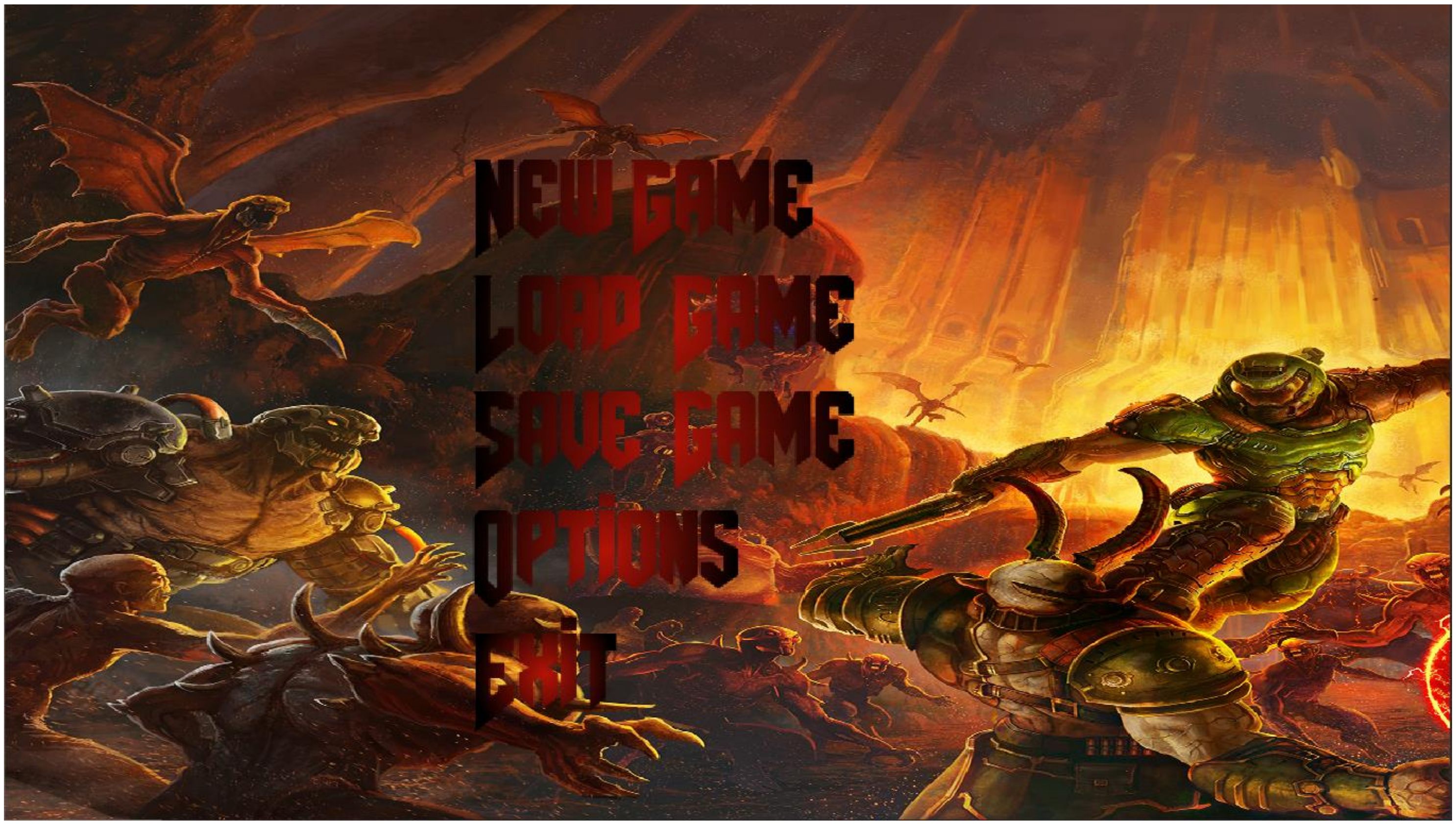
```
pip install -r /path/to/requirements.txt
```

Game overview

Different type of menus

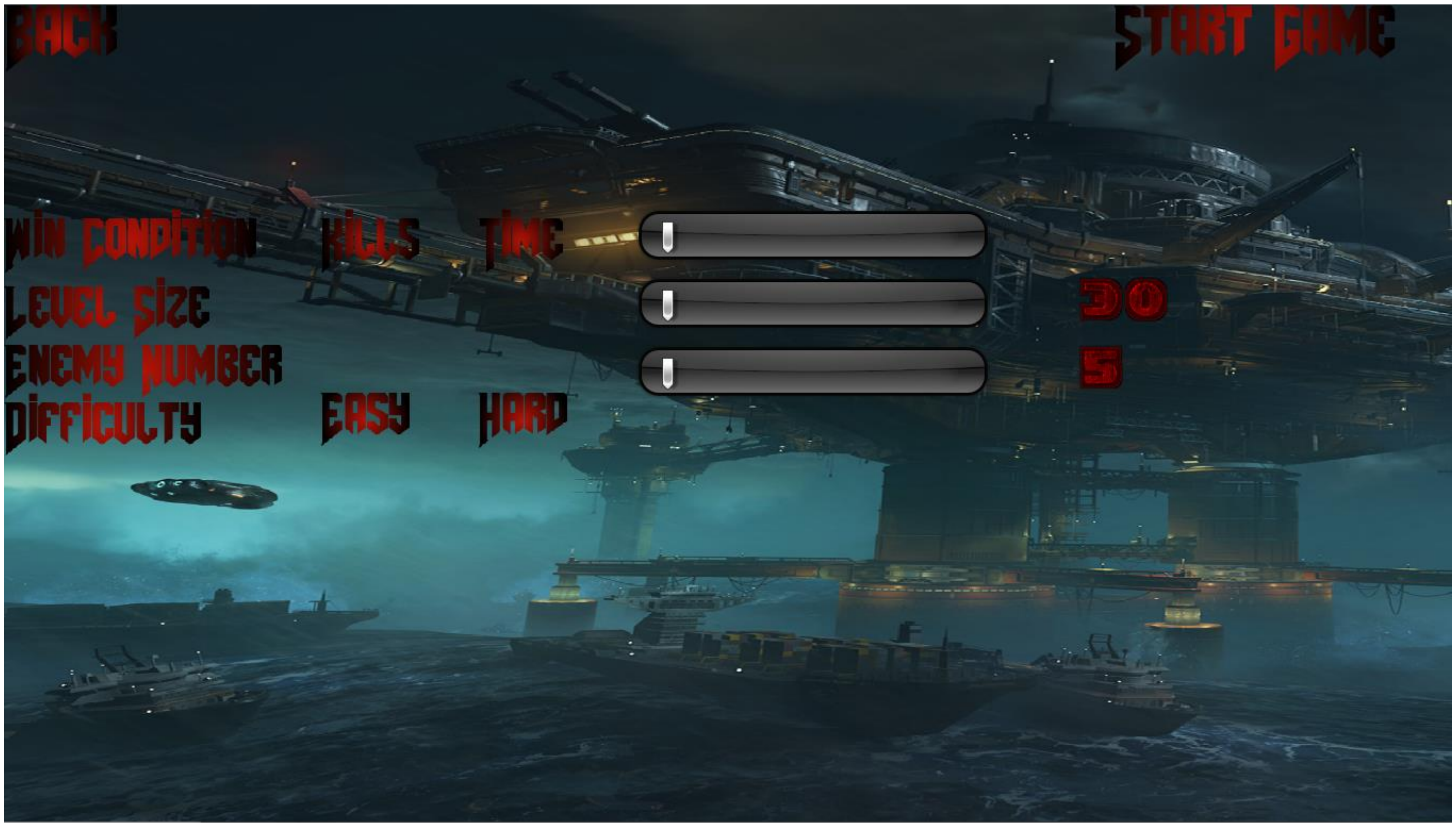
Main menu

The main menu is the first thing that the user will see.



As you can see all the different submenus are displayed. The navigation happens with the mouse and the user can choose the corresponding menu when he clicks on them. Every menu has unique moving background. If the user hovers over one of them the text will appear larger and if clicks on it, he/she will be directed to the desired submenu.

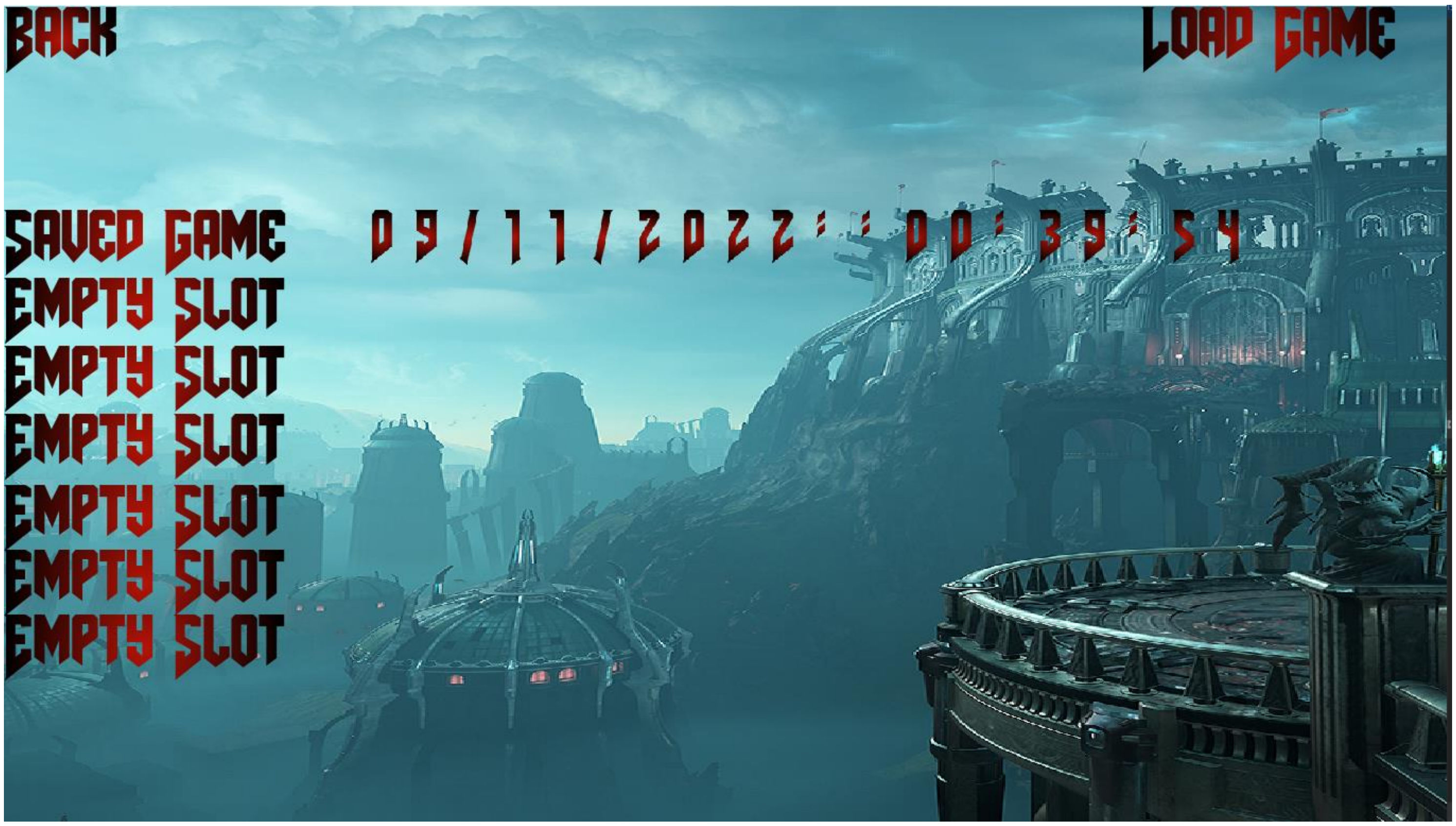
New game menu



This is the new game menu. There are different types of options that the user can choose before creating his new game. He can choose the winning condition of the game will it be killing enemies or trying to survive for certain amount of time.

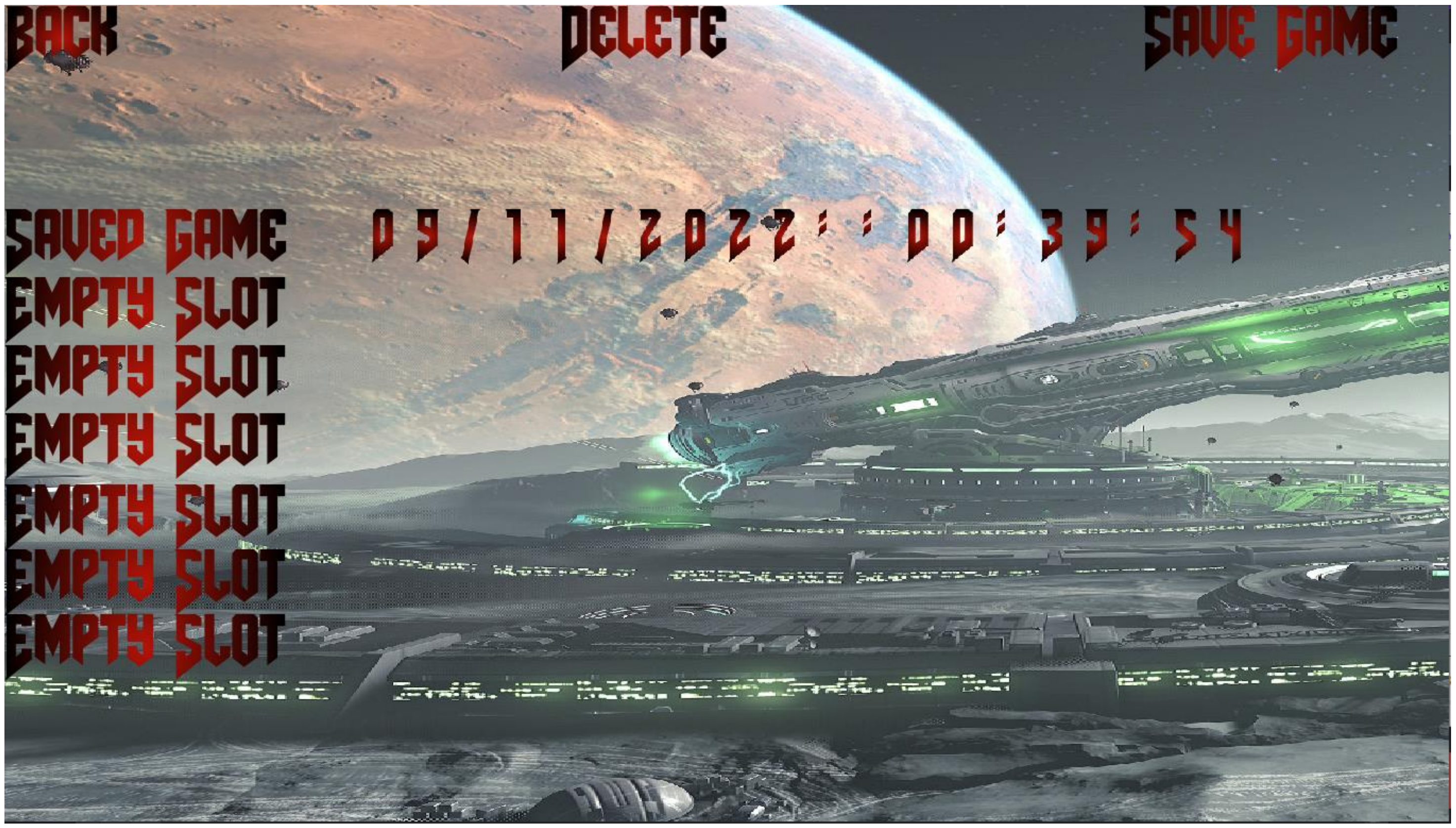
He can select what will be the generated map size. The maximum number of enemies that can spawn on the map at once and the difficulty level of the game. For choosing the different options when the player clicks on them, they will become larger, thus indication they are selected. If the user does not wish to choose any of those options – the default once will be loaded. When the player presses start game, the game will load and start. For going back to main menu the Back button must me clicked.

Load game menu



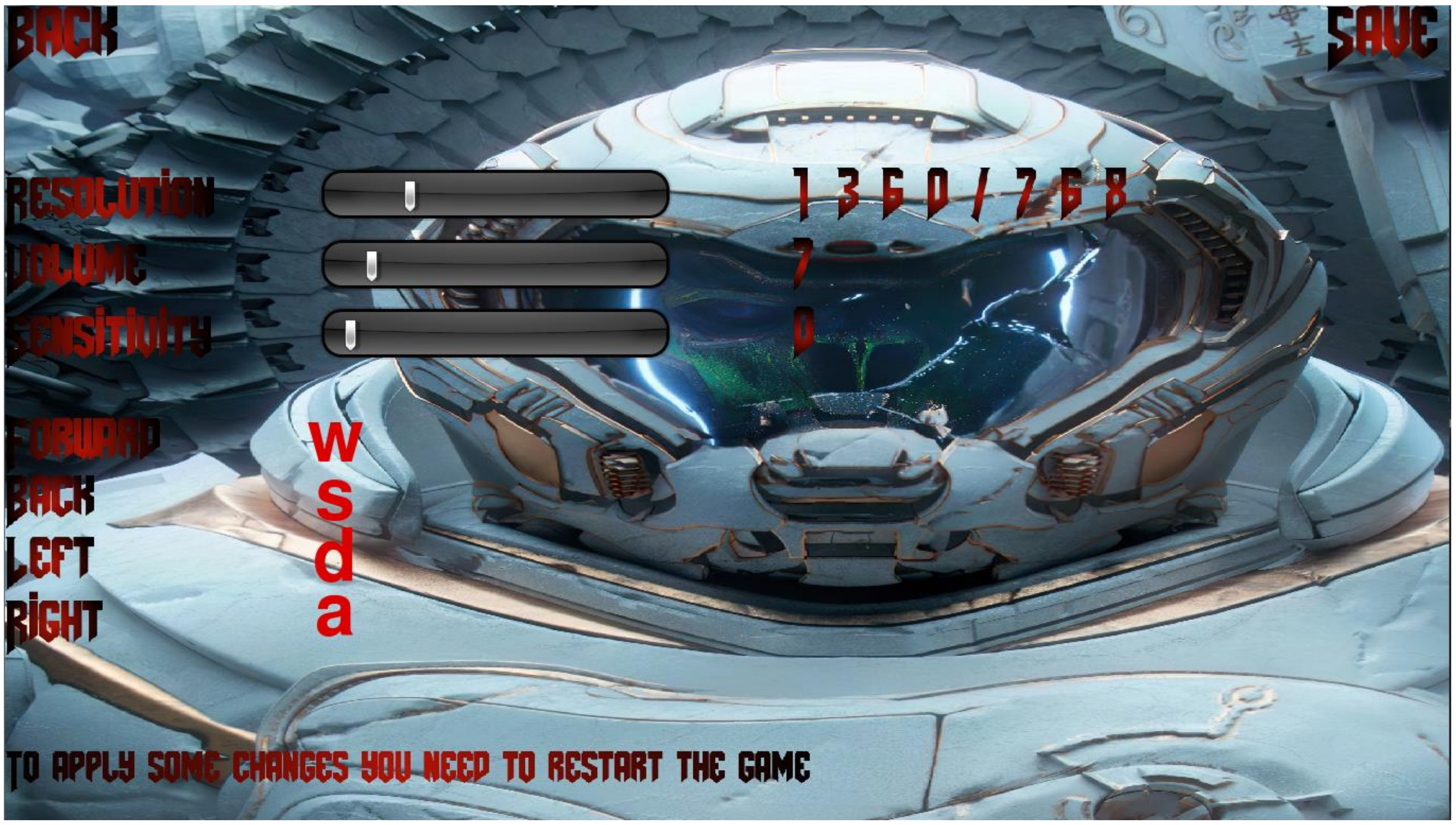
This is the loading game menu. If there is a saved game it will appear as Saved Game and after that the date the game was saved, otherwise it will appear as an empty slot. When the user hovers or clicks on empty slot or saved game the selected item will become larger indicating that it is selected. If the player wants to play the selected save, he/she must click Load Game button on top. If the user wants to go back, he/she should click the Back button.

Save game menu



This is the save game menu. The user will see all his saved game as Saved Game and if there are no saves empty slot will appear like shown above. There is functionality to delete a save after you have chosen it. Again the player can over ride saved games or choose a new slot to save his/her progress so far, by clicking on save game.

Options menu



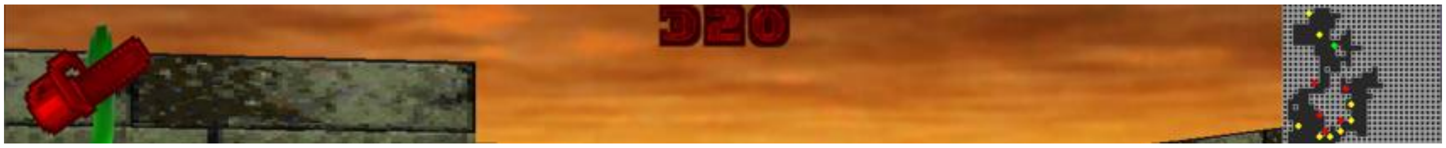
1360/768
7
0
w
s
d
a

TO APPLY SOME CHANGES YOU NEED TO RESTART THE GAME

This is the options menu. From here the user can choose between various options like changing the resolution, volume, mouse sensitivity and can assign key binding to the movement keys. The changes will be saved when the player press the save button. It is important to mention that when changing the resolution, to apply the changes the game needs to be restarted. The changes are saved so that even when the player exits the game and re-enters it later, the options that he/she saved will be preserved.

Win conditions

Time based win condition



Time based win condition is how much time the player can survive. If he/she survives for a certain amount of time, he/she wins. The amount of time can be selected from the new game menu – it is in seconds. As soon the player starts the game a timer will appear on the game screen if the timer reaches 0 and the player is still alive – the player wins the game.

Enemy kills win condition

If the desired win condition is winning the game based on the number of enemies that the player has killed. In bottom of the screen there is frag tab. It measures the enemies that have been slayed by the player. When they reach the specified by the user amount the game will be won. If the user dies before reaching that amount, the game will be lost.

Difficulty

Easy difficulty

If the player wants easier experience, he/she can choose easy difficulty in the new game menu. The easy difficulty is the default difficulty. When it is chosen the enemies are weaker – have less health and deal less damage.

Hard difficulty

If the player wants to challenge themselves, the hard difficulty is the option to go. All the enemies have greater health and deal more damage to the player. The loot drop from enemy kills is also decreased.

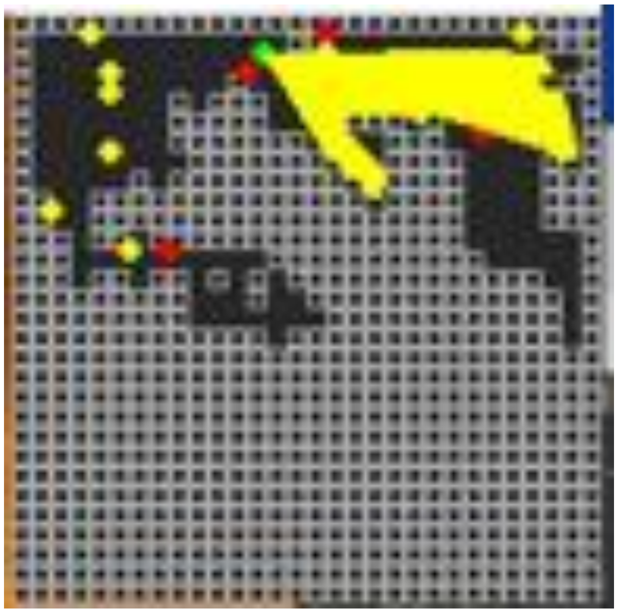
Character

Character controls

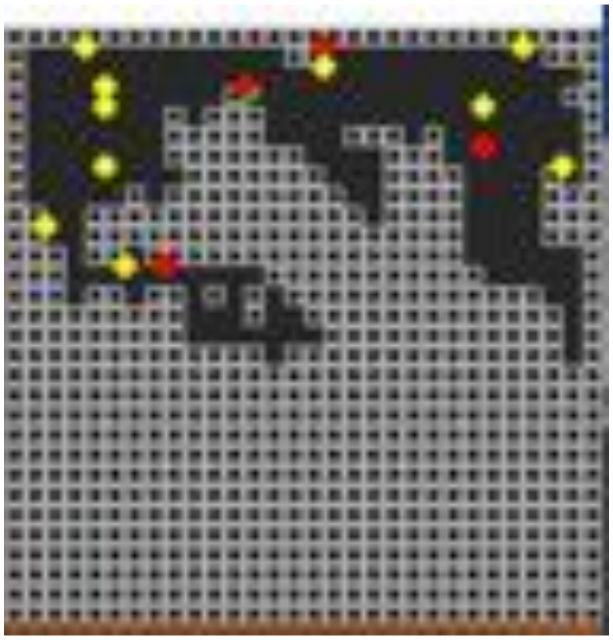
The player movement is pretty straight forward. The character can move forwards, backwards, left and right with the assigned keyboard keys in the options. The weapon changes are done with the mouse wheel or 1,2,3 – keys.

To look around the player is using the mouse and for shooting left mouse button. If you the player wants to toggle ray casting on or off on the mini map, he/she should press “r”.

Ray casting on:



Ray casting off:



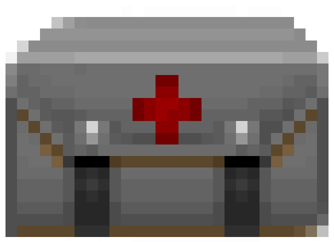
Health and armor

The character has 400 health points and 400 armor points. When the armor points are depleted, the player starts losing health. If the characters health goes to 0 the game is over and lost. There 2 methods from which the player can increase his health. Health regeneration when nobody is shooting at him or health packs that can be found in the world via loot drops from enemies or from loot boxes. The only way to regenerate armor is to pick the armor item.

The health regeneration is 1 health point per second and a half. When a health pack is picked the regenerated health is 30 points.

The armor regeneration when armor pack picked is 20 points.

Health pack:



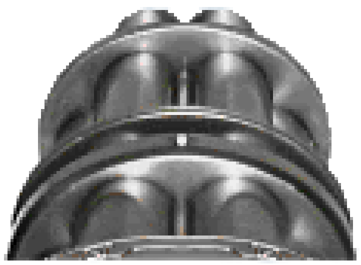
Armor pack:

Weapons

There are three different kinds of weapons in the game, each has its unique ability. The chainsaw:

Damage deal 10 hp points per frame. Which is very powerful, however the drawbacks are the fuel consumption and the relatively small range that the weapon has.

The next weapon is the mini gun:

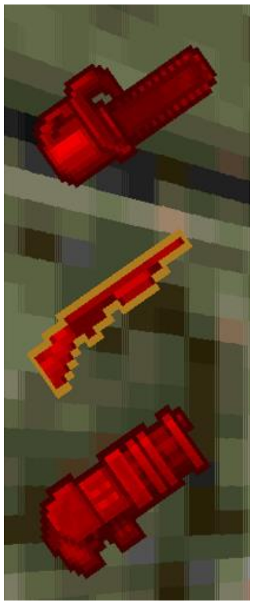


It deals smallest amount of damage compared to other weapon types, however it compensates with big fire rate and range. The damage is 5 health points every second.

And the last weapon is the shotgun:

The weapon is slow due to its reloading animation however it has great fire power – 100 health points.

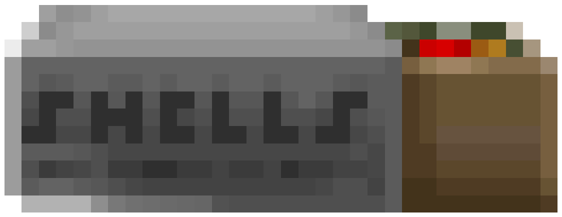
As we mentioned above the weapons can be selected with mouse scroll or the keyboard buttons 1,2,3. In the upper left corner of the screen the player can see which weapon is selected:



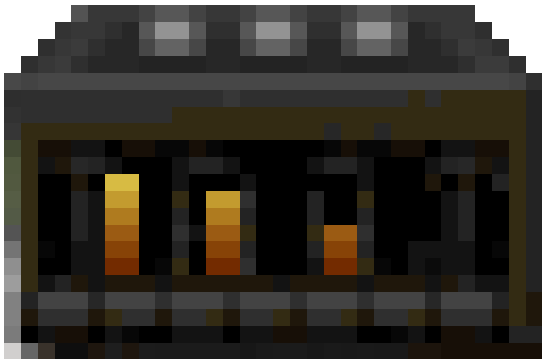
Ammo

With the three different weapon types, come the three different ammo types of course.

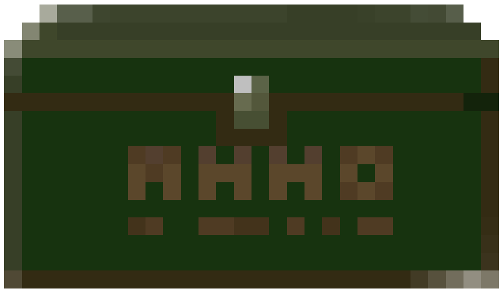
Shotgun ammo:



Chainsaw fuel:

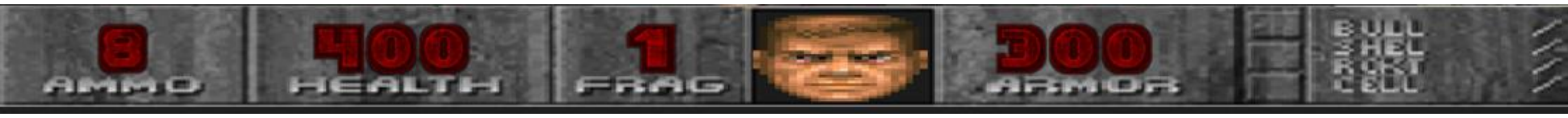


Minigun ammo:



HUD

In the HUD all the necessary game information is displayed. How many health points the player has, the armor points, ammo remaining for the chosen weapon. How many enemies where killed.



Enemies

Enemy types

There are two different enemy types. Each has its own health and damage dealing. The cyber demon enemy type is the stronger one.

Soldier enemy type:

Cyber demon enemy type:

Enemy movement

The movement of the NPCs is directed by their sight of the player using one line ray casting. If the player hides from the enemy behind a wall or obstacle the enemy pathfinding algorithms are triggered. – either A start or BSF. A star is the default algorithm used. If the enemy does not have direct sight of the player and haven’t spotted him/her previously they stay in idle state.

Enemy world placement

NPCs are randomly placed around the map. Their number depends on the maximum enemy restriction option in the start game menu. When an enemy dies it respawns again on the map. This technique can increase the difficulty of the game and make pretty interesting gameplays.

Interactions with the player and drops

Basically, seek and destroy. Their only purpose is to kill the player.

When an NPC is killed by the user, occasionally they will drop ammo, health or armor. The selection of what to drop is decided on random.

Loot boxes

(Loot barrels)

Player interactions

The only interaction that they have with the player is to get shot. When they are shot by the character, they drop random item – health, armor or ammo.

World placement

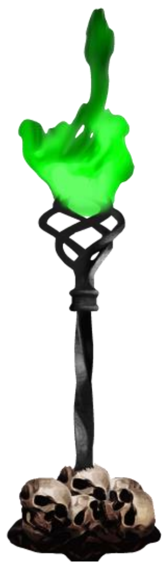
There are fixed amounts of loot barrels every game deployed in the world at random locations. If the barrel is destroyed it does not respawn again. The number of them is 10 per world. This is done to increase the level of difficulty as the player needs to manage those resources if he/she wants to survive for longer periods of time.

Drops

They drop the same loot as the NPCs when they are killed: Health, armor, minigun ammo, chainsaw fuel, shotgun ammo. However, the drop rate is 100 percent.

Static objects

Overview



At the moment there is only one static object that is animated. The green light. Its functionality is purely decorative.

World Placement

The placement is random in the map. Just like the loot barrels.

World

World generation

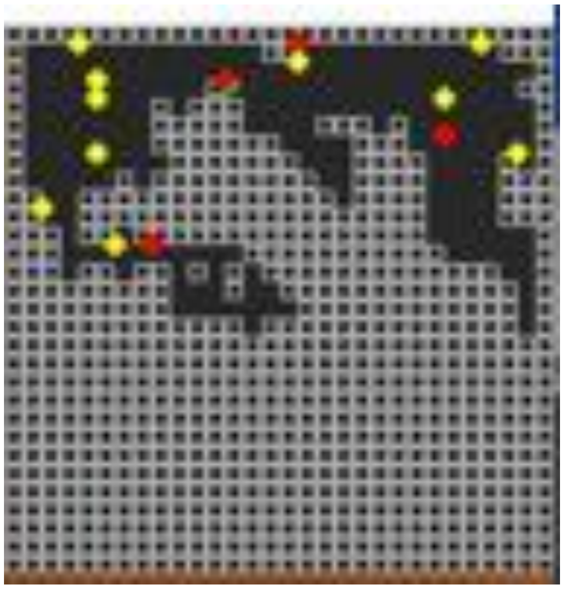
The world is a random generated tile map, using 0 for walkable path and 1 for wall. The generation of the world is done by the simple random walk algorithm and it is refined by A star algorithm. Basically, the random walk algorithm makes a few hundred maps and the most complex one is choose based on the length of the A star algorithm output.

Textures

At the current moment there is only one texture that is used in the map generation process. The floor is just dark grey color. And the sky is a single image.

Mini map

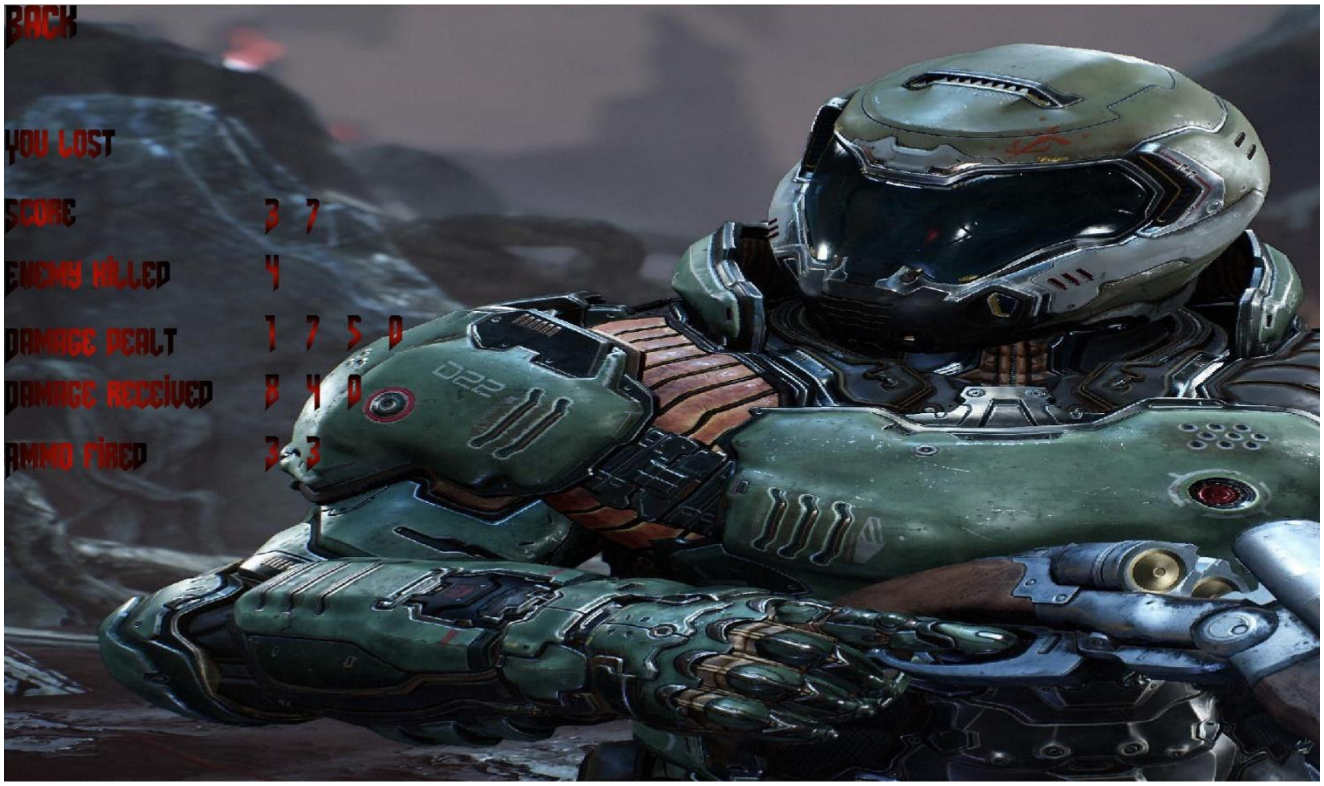
Overview



The mini map is used for better bird eye view of the current overall situation. The green dot on the map represents the player. The yellow dots represent the loot barrels and the red dots represent the enemy. It is placed on the right top corner of the screen.

End game

End game screen overview



YOU LOST

SCORE	3 7
ENEMY KILLED	4
DAMAGE DEALT	1 7 5 0
DAMAGE RECEIVED	8 4 0
AMMO FIRED	3 3

When the game ends the end game screen will show with some statistics about the game. If the player has won or lost the game is written on top. The user can see his/her final score, total enemy kills, total damage delt, total damage received and ammo fired. The back button returns the user to the main menu.

CHAPTER ONE

WOLFENSTEIN COPY

1.1 button module

class `button.` Button(*surface*, *path*, *scale*, *transform_scale_hover*, *option=False*, *action_lock=False*)

Bases: `object`

Button class

Class that makes easy placement of buttons on the screen. Build specifically for the menu.

action()

Action function of class button

It that returns True if button is clicked. If action_lock - it always return true

Returns

True or False

Return type
bool

draw(*x*, *y*)

Drawing function of the button.

Placing the rect on the coordinates. Takes as arguments x and y which are the positions

Parameters

- x (*float*) – x position on the scree
- y (*float*) – y position on the screen

Returns
None

Return type
None

property get_size

class `button`. Slider(*surface*, *pos_x*, *pos_y*, *saving=False*, *value=None*)

Bases: `object`

Slider class

Slider class that makes easy placing the sliders on the menu.

draw()

Drawing function of the slider

Returns
None

Return type
None

property get_position

Property method of class slider for getting the position of the slider

Returns
x and y position of slider

Return type
tuple

property get_slider_value

Property method of class Slider for getting the value of the slider.

Calculates the sliders value. If the number is negative returns 0. Used to determine the option value eg :mouse sensitivity, volume, screen resolution.

Returns
number - slide position on the screen

Return type
int

update()

Update function for the slider to check the mouse position

Returns
None

Return type
None

1.2 hud module

class hud.Hud(*game*)

Bases: object

Class hud

Everything that is showed on the screen like ammo, weapon selections, small player sprite animations, without the weapons and the minimap

animate_player_hud_sprite()

Animation function for the small player sprite in the middle of the hud.

Returns
None

Return type
None

draw()

Drawing function

Returns
None

Return type
None

get_images(*path*)

Function for getting the hud images and converting them to transparent

Parameters
path (*str*) – path to the image

Returns
images

Return type
deque

helper_for_hud_selected_weapon_view()

Helper function for displaying selected weapons

Returns
None

Return type
None

1.3 interactions module

class interactions.ObjectInteraction(*game*, *path='resources/sprites/interactables/barrel/0.png'*, *pos=(10.5, 5.5)*, *scale=0.5*, *shift=0.5*, *gone=False*)

Bases: *AnimatedSprite*

Class object interactions.

This clss is responsible for the loot drop objects - barrels animate_death()

Function that animates destruction of the barrel

Returns

None

Return type

None

animate_idle()

Function that animates idle of the barrel

Returns

None

Return type

None

check_if_hit()

Function that checks if the object is hit or not.

Returns
None

Return type
None

logic()

Function that deals with the logic of the barrel

Returns
None

Return type
None

loop_drop()

Function that deals with loot drop logic

Returns
None

Return type
None

property map_pos

property method for objects position.

Returns
barrels position

Return type
int

update()

Update method.

Returns
None

Return type
None

interactions.random() → x in the interval [0, 1).

interactions.ray_cast_player_npc(*player_position*, *player_map_position*, *self_map_position*, *game_map*, *theta*)

Function for raycasting,

It checks if the player has visual of the object. And determines if he can shoot it.

Parameters

- player_position (*tuple*) – The position of the player
- player_map_position (*tuple*) – The map tile position of the player
- self_map_position (*tuple*) – The map position of the object
- game_map (*dict*) – The game map
- theta (*float*) – theta angle

Returns

True or False

Return type
bool

1.4 main module

class `main`. Game

Bases: `object`

The main class of the game.

Here all the classes from the different files are initialized. The main game loop runs here

check_events()

Check event method.

Checking the event loop for mouse clicks, movement etc.

Returns
None

Return type
None

draw()

Drawing method.

All the drawing logic is here.

Return

s

Return

type

menu()

Function for calling the main menu at the start of the app.

Returns
None

Return type
None

new_game()
Function for making instances of classes that are in the files.

Returns
None

Return type
None

run()
Main run loop.

Returns
None

Return type
None

update()

Update function

All the update logic is here

Returns
None

Return type
None

`main`. random() → x in the interval [0, 1).

1.5 main_menu module

class `main_menu`. FinalScore(*game*)

Bases: `object`

Class FinalScore

back_main_menu()

Function for calling the main menu.

Returns
None

Return type
None

check_events()

Function for event loop.

Returns
None

Return type
None

draw()

Function for drawing the stats game menu

Returns
None

Return type

None

draw_background()

Function for drawing the moving background

Returns
None

Return type
None

draw_numbers(*number*, *width*, *height*)

Function for drawing the numbers.

Parameters

- number (*int*) – number
- width (*int*) – width of the number
- height (*int*) – height of the number

Returns
None

Return type
None

get_textures(*path*, *scale*)

Function for getting the images

Parameters

- path (*str*) – Path to the image
- scale (*float*) – Scale of the image

Returns
image

Return type
pygame.image

run()

Function for game stats menu running loop.

Returns
None

Return type
None

update()

Function for updating.

Returns
None

Return type
None

class
main_menu. LoadGameMenu
(*game*)

Bases: object

Load game class.

back_main_menu()

Function for calling the main menu.

Returns
None

Return type
None

check_events()

Function for event loop.

Returns
None

Return type

None

choose_slot()

Function for choosing the slot.

When user clicks on it, the slot is selected and transformed to bigger scale.

Returns

None

Return type

None

draw()

Function for drawing the load game menu

Returns

None

Return type

None

draw_background()

Function for drawing the moving background

Returns

None

Return type

None

draw_date(*height*, *data*)

Function for drawing the date when game is saved.

Parameters

- height (*float*) – Placement on the screen
- data (*datetime*) – date

Returns

None

Return type
None

get_textures(*path, scale*)

Function for getting the images

Parameters

- path (*str*) – Path to the image
- scale (*float*) – Scale of the image

Returns
image

Return type
pygame.image

load_game()

Function for loading the game

Returns
None

Return type
None

run()

Function for save menu running loop.

Returns
None

Return type
None

show_load_games()

Function for showing the saved game slots and date

Returns
None

Return type
None

show_loaded_game_slots()

Function for assigning saved game buttons to the empty slot list.

Returns
None

Return type
None

update()

Function for updating.

Returns
None

Return type
None

class main_menu.MainMenu(*game*)

Bases: `object`

Main menu class

check_events()

Function for event loop.

Returns
None

Return type
None

draw()

Function for drawing the main menu

Returns

None

Return type

None

draw_background()

Function for drawing the moving background

Returns

None

Return type

None

exit_game()

Function for exiting the game.

Returns

None

Return type

None

load_game()

Function for calling the load game menu.

Returns

None

Return type

None

options()

Function for calling the options menu.

Returns

None

Return type

None

resume_game()

Function for pausing the game.

Returns
None

Return type
None

run()

Function for main menu running loop.

Returns
None

Return type
None

save_game()

Function for calling the save game menu

Returns
None

Return type
None

start_new_game()

Function for calling the new game menu.

Returns
None

Return type
None

update()

Function for updating.

Returns
None

Return type
None

class `main_menu`. NewGame(*game*)

Bases: `object`

NewGame class.

back_main_menu()

Function for calling the main menu.

Returns
None

Return type
None

check_events()

Function for event loop.

Returns
None

Return type
None

draw()

Function for drawing the new game menu

Returns
None

Return type
None

draw_background()

Function for drawing the moving background

Returns

None

Return type

None

draw_numbers(*number*, *width*, *height*)

Function for drawing numbers

Parameters

- number (*int*) – Number to be drawn.
- width (*int*) – Width of the number
- height (*int*) – Height of the number

Returns

None

Return type

None

enemy_count_options()

Function for enemy count.

The user can choose the enemy count before starting the new game.

Returns

None

Return type

None

game_difficulty_options()

Function for choosing the game difficulty

Returns

None

Return type

None

get_textures(*path*, *res=(256, 256)*)

Function for getting the digits.

Parameters

- path (*str*) – path to the image
- res (*list*) – predefined parameter

Returns
None

Return type
None

map_generation_options()

Function for map generation option.

The user can choose the map size before starting the new game.

Returns
None

Return type
None

run()

Function for new game menu running loop.

Returns
None

Return type
None

start_new_game()

Function for starting the new game.

Returns
None

Return type
None

update()

Function for updating.

Returns
None

Return type
None

win_condition_options()

Function to choose the winning condition of the game.

Returns
None

Return type
None

class `main_menu.`Options(*game*)

Bases: `object`

Class Options

back_main_menu()

Function for calling the main menu.

Returns
None

Return type
None

check_events()

Function for event loop.

Returns
None

Return type
None

draw()

Function for drawing the options game menu

Returns

None

Return type

None

draw_background()

Function for drawing the moving background

Returns

None

Return type

None

draw_inputs()

Function for drawing the user inputs

Returns

None

Return type

None

draw_inputs_helper(*key_binding*, *width*, *height*)

Helper function of draw inputs.

Parameters

- key_binding (*int*) – The key that needs to be bind.
- width (*int*) – Width position
- height (*int*) – Height position

Returns

None

Return type

None

draw_numbers(*number, width, height*)

Function for drawing the numbers.

Parameters

- number (*int, str*) – number
- width (*int*) – width of the number
- height (*int*) – height of the number

Returns
None

Return type
None

get_textures(*path, scale*)

Function for getting the images

Parameters

- path (*str*) – Path to the image
- scale (*float*) – Scale of the image

Returns
image

Return type
pygame.image

key_bindings()

Function for user keybindings.

Returns
None

Return type
None

mouse_sensitivity_option()

Function for choosing the mouse sensitivity.

Returns
None

Return type
None

resolution_option()

Function for choosing the resolution of the screen

Returns
None

Return type
None

run()

Function for options menu running loop.

Returns
None

Return type
None

save_options()

Function for saving the user chosen options.

Returns
None

Return type
None

update()

Function for updating.

Returns
None

Return type
None

volume_option()

Function for choosing the volume.

Returns
None

Return type
None

class main_menu. SaveGameMenu(*game*)

Bases: object

Save menu class.

back_main_menu()

Function for calling the main menu.

Returns
None

Return type
None

check_events()

Function for event loop.

Returns
None

Return type
None

choose_slot()

Function for choosing the slot.

When user clicks on it, the slot is selected and transformed to bigger scale.

Returns
None

Return type
None

delete_saved_games()

Function for deleting the saved games.

Returns
None

Return type
None

draw()

Function for drawing the save game menu

Returns
None

Return type
None

draw_background()

Function for drawing the moving background

Returns
None

Return type
None

draw_date(*height*, *data*)

Function for drawing the date when game is saved.

Parameters

- height (*float*) – Placement on the screen
- data (*datetime*) – date

Returns
None

Return type
None

get_textures(*path*, *scale*)

Function for getting the images

Parameters

- path (*str*) – Path to the image
- scale (*float*) – Scale of the image

Returns
image

Return type
pygame.image

run()

Function for save menu running loop.

Returns
None

Return type
None

save_game()

Function for saving the game.

Returns
None

Return type
None

show_saved_game_slots()

Function for assigning saved game buttons to the empty slot list.

Returns
None

Return type

None

show_saved_games()

Function for showing the saved games.

Returns

None

Return type

None

update()

Function for updating.

Returns

None

Return type

None

1.6 map module

class map.Map(*game*)

Bases: `object`

Class Map

draw()

Function for drawing the minimap on the screen

Returns

None

Return type

None

get_map()

Function for getting the map.

Appends in a dictionary all the map values that are walkable.

Returns
None

Return type
None

load_minimap()

Function for loading an existing map.

Returns
None

Return type
None

1.7 npc module

class npc.CyberDemonNPC(*game, path='resources/sprites/npc/cyber_demon/0.png', pos=(11.5, 6.0), scale=1.0, shift=0.04, animation_time=210*)

Bases: *NPC*

class npc.NPC(*game, path='resources/sprites/npc/soldier/0.png', pos=(10.5, 5.5), scale=0.6, shift=0.38, animation_time=180, gone=False*)

Bases: *AnimatedSprite*

Class npc.

animate_death()

Function for animating death.

Returns
None

Return type
None

animate_pain()

Function for animating pain

Returns
None

Return type
None

attack()

Function for attacking the player with some random accuracy.

Returns
None

Return type
None

check_health()

Function for checking NPC health

Returns

None

Return type

None

check_hit_in_npc()

Function for checking if player has hit NPC with different weapons

Returns

None

Return type

None

check_wall(*x*, *y*)

Helper for the check wall collision
function.

Parameters

- x (*int*) – x coordinates of npc.
- y (*int*) – y coordinates of npc.

Returns
x and y if not in the world map

Return type
tuple

check_wall_collision(*dx*, *dy*)

Checking for wall collision. Using the size of the npc sprite

Parameters

- dx (*float*) – next x coordinate position of npc.
- dy (*float*) – next y coordinate position of npc.

Returns
None

Return type
None

loop_drop()

Function for dropping loot logic.

Checks if the NPC is alive and if the drop_loot is True If drop is bigger than random it drops random selected item.

Returns
None

Return type
None

property map_pos

Property function to return npc position on map

Returns
x and y

Return type
tuple

movement()

Function for NPC movement

1If the pathfinding library is installed the movement is done by A star algorithm, else BFS

:return None :rtype: None

path_find(*start*, *goal*, *mini_map*)

Helper function for A star algorithm of the path finding library

Parameters

- start (*tuple*) – map tile position of npc
- goal (*tuple*) – map tile position of player
- mini_map (*list*) – take s the game map

Returns

last list value

Return type

tuple

ray_cast_player_npc()

Raycasting for NPC.

One line raycasting between NPC and player. Is used for shooting logic and for pathfinding logic

Returns

True or False

Return type

Bool

run_logic()

Function for npc run logic.

Returns

None

Return type

None

update()

Update method.

Returns

None

class npc.SoldierNPC(*game, path='resources/sprites/npc/soldier/0.png', pos=(10.5, 5.5), scale=0.6, shift=0.38, animation_time=180*)

Bases: *NPC*

npc.random() → x in the interval [0, 1).

1.8 object_handler module

class object_handler.ObjectHandler(*game*)

Bases: `object`

Class for handling the in game object. Animated sprites, npc, loot boxes.

`add_interactables`(*sprite*)

Function for adding interactable objects to the interactables list

Parameters
sprite – sprite object

Returns
None

`add_npc`(*npc*)

Function for adding npc objects to the npc list.

Parameters
npc – npc object

Returns

None

add_pick_sprites(*sprite*)

Function for adding pick sprites obejct to the pick sprites list

Parameters
sprite – sprite object

Returns
None

add_sprite(*sprite*)

Function for adding sprite objects to the sprite lists

Parameters
sprite – sprite object

Returns
None

draw()

Drawing function for drawing the npc positions and the barrel positions on the minimap

Returns
None

spawn_collectables()

Spawn function for collectable objects.

It checks if the load game parameter is true or not. If the load game boolean parameter is true then adds the collectable according to their previously saved positions If the load game boolean parameter is false, the function randomly places the collectable objects in the world

Returns
None

spawn_npc()

Spawn function for npc objects.

Adds the npc to the list on condition if the len of the npc_list is less than the desired one It checks if the load game parameter is true or not. If the load game boolean parameter is true then adds the npc according to their previously saved positions If the load game boolean parameter is false, the function randomly places the npc objects in the world

Returns
None

spawn_sprites()

Spawn function for sprite objects.

It checks if the load game parameter is true or not. If the load game boolean parameter is true then adds the sprites according to their previously saved positions If the load game boolean parameter is false, the function randomly places the sprites objects in the world

Returns
None

update()

Updating method that is spawning the npc, getting npc positions, sprite positions, pick sprites positions and updates the lists of npc and sprites.

Returns

npc_positions, npc_type_list, pick_sprites_positions, sprite_list_positions

`object_handler.` random() → x in the interval [0, 1).

1.9 path_find_bfs module

class `path_find_bfs.` PathFinding BFS(*game*)

Bases: `object`

Class PathfindingBFS

bfs(*start*, *goal*, *graph*)

Function for BFS algorithm

More info in the docs under section Explanation

Parameters

- start (*int*) – Starting position
- goal (*tuple*) – Ending position
- graph (*dict*) – The graph dict

Returns

visited dict

Return type

dict

get_graph()

Function for creating the graph for the BFS.

It is a dictionary and we are giving the position plus the next positions

Returns

None

Return type

None

get_next_nodes(*x*, *y*)

Function for defining the next positions

Returns
list of the next positions

Return type
list

get_path(*start*, *goal*)

Function for getting the path

Parameters

- start (*int*) – Start positions
- goal (*tuple*) – End of the path

Returns
last position of the path

Return type

tuple

1.10 player module

class `player.` **Player**(*game*)

Bases: `object`

Class Player

check_ammo()

Function to check if ammo is negative if it is negative make it to 0

Returns

None

Return type

None

check_game_over()

Function for checking game over condition

Returns

None

Return type

None

check_health_recovery_delay()

Function for health recovery delay

Returns

True if condition is men

Return type

bool

draw()

Function for drawing the player on the minimap

Returns

None

Return type

None

fuel_consumption_chainsaw()

Function that checks the fuel consumption of the chainsaw

Returns

None

Return type

None

get_damage(*damage*)

Funtion for taking damage.

Taking damage from NPC and taking from armor as well the armor reduces the damage rendering the damage effect and sound checking if player is alive

Parameters
damage (*int*) – damage taken

Returns
None

load()

Return type
None

Loading function of the saved player parameters

Returns
None

Return type
None

property map_pos

Function for getting the tile position the player is

Returns
x and y

Return type
tuple

minigun_ammo_consumption()

Function that checks the ammo consumption of the minigun

Returns
None

Return type
None

mouse_control()

Function for mouse movement

Returns
None

Return type

None

movement()

Function for player movement

Fist we calculate the increments because we are on 2D using the functions of sin and cos Used for the player direction angle. And getting out to the menu

Returns
None

Return type
None

place_player()

Function for random player placement

Returns
player x and y

Return type

tuple

property pos

Function for getting the player position

Returns

x and y

Return type

tuple

recover_health()

Function for health recovery

Returns

None

Return type

None

score()

Function for calculating the total score for the end game

Returns

None

Return type

None

single_fire_event(*event*)

Function for shooting logic.

When chainsaw selected to shoot until mouse button up, when shotgun selected shoot and animate, when minigun selected shoot like chainsaw and ammo restrictions

Parameters

event (*event*) – event type

Returns

None

Return type
None

update()

Updating function

Returns
None

Return type
None

wall_check(*x*, *y*)

Checking where are the walls in the map

Parameters

- x (*float*) – Player x position
- y (*float*) – Player y position

Returns

x and y

Return type

float

wall_collision(*dx*, *dy*)

Function for checking wall collision

Parameters

- dx (*float*) – x position
- dy (*float*) – y position

Returns

None

Return type

None

weapon_selection(*event*)

Function for weapon selection based on 1,2,3 keys and mouse wheel

Parameters

event (*event*) – event pygame

Returns

None

Return type

None

weapon_selector()

Helper function for weapon selection

Returns

None

Return type

None

win_condition()

Function for win condition on npc killed or on time passed

Returns
None

Return type
None

1.11 random_map_generation module

`random_map_generation.` drunken_walk()

Very simple implementation of the random walk algorithm

Returns
level, start_coordinate, end_coordinate

Return type
list

random_map_generation. **evaluate_levels**(*levels*)

Function that evaluated the level created

The path needs to be the longest gor the level to be selected as playable

Parameters
levels (*list*) – Map level

Returns
Evaluation scores

Return type
list

random_map_generation. **generate_best_level**(*number_of_levels*)

Function for choosing the best level

Takes the longest path to play if there is the pathfinding library else we return the first generated level

Parameters
number_of_levels (*int*) – Number of levels

Returns
Best level

Return type
list

random_map_generation. **get_path**(*start*, *goal*, *random_map*)

Helper function for calling the A star algorithm.

Parameters

- start (*tuple*) – starting coordinates
- goal (*tuple*) – Ending coordinates
- random_map – The generated map

:type random_map:list :return: path
:rtype: list

1.12 raycasting module

class `raycasting`. RayCasting(*game*)

Bases: `object`

Class RayCasting

get_objects_to_render()

Functions for getting the objects that need to be rendered

This function transforms the images to the output of the ray_cast function. Appends them to a list and sends them to render class

Returns
None

Return type
None

ray_cast()

Ray casting function.

More info in docs under Explanation.

Returns
None

Return type
None

update()

Update function for the class

Returns
None

Return type
None

1.13 renderer module

class renderer.ObjectRenderer(*game*)

Bases: `object`

Class ObjectRenderer

draw()

Drawing function

Returns
None

Return type
None

draw_background()

Function for drawing the background.

Sky image with some offset to the player position

Returns
None

Return type
None

draw_frag_counter()

Function for showing the player kills on the screen

It enumerates the int number that is given and appends the digit number to the given height and width

Returns
None

Return type
None

draw_player_ammo()

Function for showing the player ammo on the screen

It enumerates the int number that is given and appends the digit number to the given height and width

Returns
None

Return type
None

draw_player_armor()

Function for showing the player armor on the screen

It enumerates the int number that is given and appends the digit number to the given height and width

Returns
None

Return type
None

draw_player_health()

Function for showing the player health on the screen.

It enumerates the int number that is given and appends the digit number to the given height and width

Returns
None

Return type
None

draw_timer()

Function for drawing the timer if win condition by time is True

Returns
None

Return type
None

game_over()

Function for drawing loosing screen

Returns
None

Return type
None

static get_textures(*path*, *res=(256, 256)*)

Static method for getting images and converting them

Parameters

- path (*str*) – path to image
- res (*tuple*) – resolution

Returns
transformed images

Return type

pygame image

load_wall_textures()

Function for loading the textures of the wall

Returns

dict of the textures

Return type

dict

player_damage()

Function for drawing when player is damaged

Returns

None

Return type

None

render_game_object()

Function for rendering the game textures from the ray casting module

Returns

None

Return type

None

win()

Function for drawing win screen

Returns

None

Return type

None

1.14 save_and_load_game module

class save_and_load_game. LoadGame (*game=None*)

Bases: `object`

Class load game

load_options ()

Function for loading the saved options

Returns

None

Return type

None

open_game_data (*name*)

Function for loading the game data

Parameters

name (*str*) – name

Returns

None

Return type

None

open_menu_saving_data()

Function for loading the save and load menu if there are saved games

Returns

dict of saved menu state

Return type

dict

class `save_and_load_game`. SaveGame(*game*)

Bases: `object`

delete_save_game(*i*)

Function for deleting the saved games

Parameters

i (*int*) – number of game to delete

Returns

None

Return type

None

save_game_data(*i*)

Function for saving the games

Parameters

i (*int*) – saved game number

Returns

None

Return type

None

save_menu_state(*i*, *date*)

Function fot save and load menus status. For showing the saved games.

Parameters

- i (*int*) – number
- date (*date*) – date of the file

Returns
None

Return type
None

save_options(*game_options_data*)

Function for saving the game options

Parameters
game_options_data (*dict*) – dict of game options save

Returns

None

Return type

None

take_game_data()

Function for taking the game data

Taking all the data that is needed to make a successful save

Returns

None

Return type

None

1.15 settings module

1.16 sound module

class sound. Sound(*game*)

Bases: object

Class Sound

update()

Updating function

Returns

None

Return type

None

1.17 sprite_object module

class sprite_object. AnimatedSprite(*game, path='resources/sprites/animated_sprites/green_light/0.png',* *pos=(11.5, 3.5), scale=0.8, shift=0.16, animation_time=120,gone=False*)

Bases: *SpriteObject*

AnimatedSprite class

animate (*images*)

Function that animates the sprite

Parameters

images (*pygame.images*) – Sprite images

Returns

None

Return type

None

check_animation_time()

Function for checking animation time

Returns
None

Return type
None

get_images(*path*)

Function for getting the sprite images.

Parameters
path (*str*) – Path to the image folder

Returns
image list

Return type
list

property map_pos

Properyty method of the class

Returns
Position of the sprite

Return type
int

update()

Updating function

Returns
None

Return type
None

class `sprite_object`. SpriteObject(*game, path='resources/sprites/static_sprites/candlebra.png', pos=(10.5, 3.5), scale=0.7, shift=0.27, pick=False, type=None, gone=False*)

Bases: `object`

Base class SpriteObject.

get_sprite()

Function for getting the sprite to player.

More information section Explanation in docs

Returns
None

Return type
None

get_sprite_projection()

Function for getting the

sprite projection. More

info in section

Explanation in docs

Returns
Bone

Return type

None

pick_object()

Pick object function.

Function that checks if the sprites is picked if it can be picked and safeguard maximum quantity of each item

Returns

None

Return type

None

update()

Updating function

Returns

None

Return type

None

1.18 weapons module

class weapons.Chainsaw(*game*, *path='resources/sprites/weapon/chainsaw/0.png'*, *scale=3*, *animation_time=90*)

Bases: *AnimatedSprite*

Class Chainsaw

animate_vruum_vruum()

Animation function

Returns

None

Return type

None

draw()

Drawing function

Returns
None

Return type
None

update()

Updating function

Returns
None

Return type
None

class weapons.Minigun(*game*, *path='resources/sprites/weapon/minigun/0.png'*, *scale=3*, *animation_time=90*)

Bases: *AnimatedSprite*

Class Minigun

animate_minigun()

Animation function

Returns

None

Return type

None

draw()

Drawing function

Returns

None

Return type

None

get_images(*path*)

Function to get the images when minigun is not firing or out of ammo

Parameters

path (*str*) – path to images

Returns

images list

Return type

list

update()

Updating function

Returns

None

Return type
None

class weapons.Shotgun(*game*, *path='resources/sprites/weapon/shotgun/0.png'*, *scale=0.4*, *animation_time=90*)

Bases: *AnimatedSprite*

animate_shot()

Animation function

Returns
None

Return type
None

draw()

Drawing function

Returns
None

Return type

None

update()

Updating function

Returns

None

Return type

None

Code Explanation

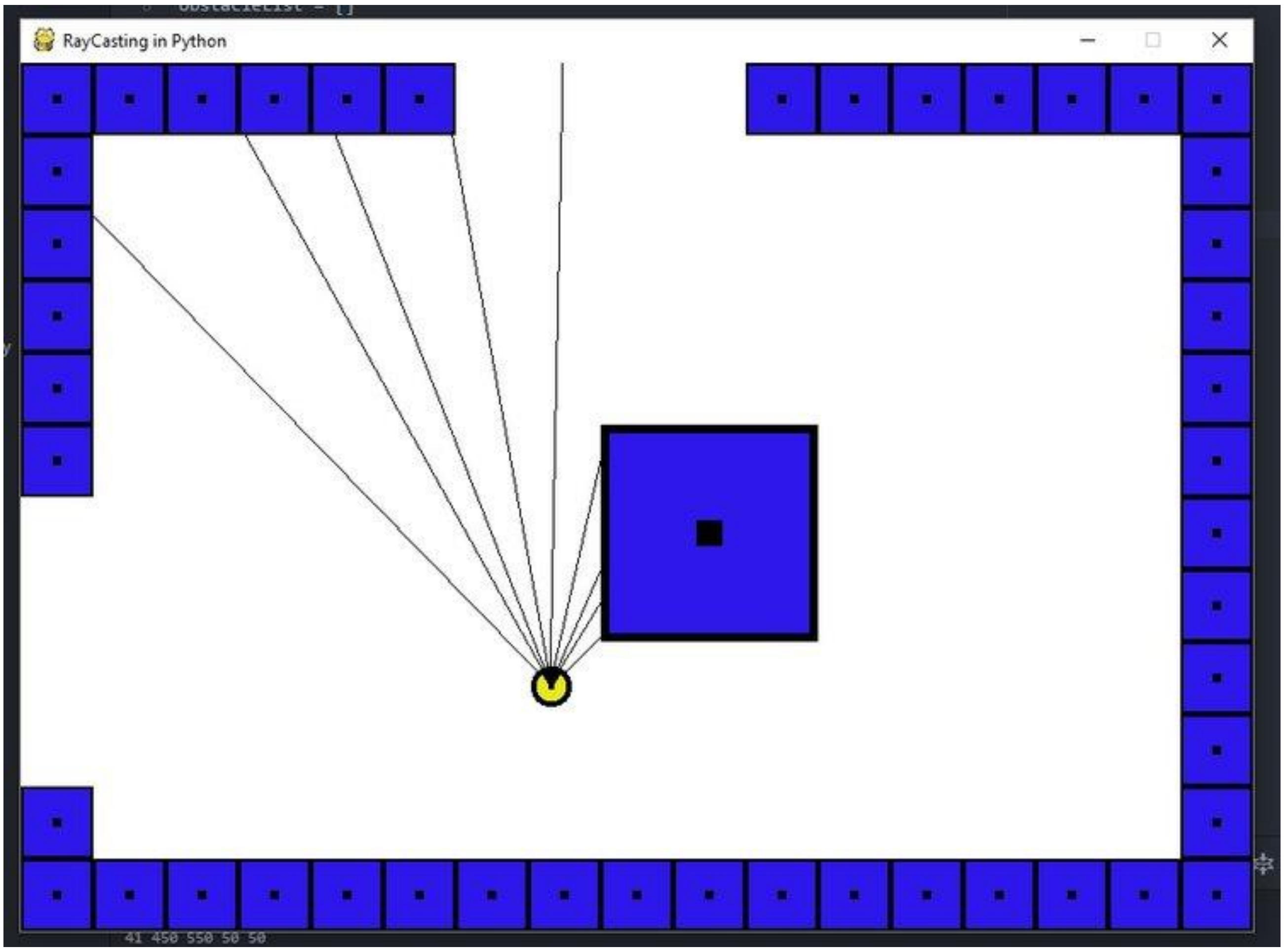
Ray casting

Introduction

In this project the ray casting algorithm is basically the game engine. Ray casting has a lot uses in modern game development such as bullet tracing, collisions and much more. However, in here we are going to view it as engine for our game.

So, what is ray casting

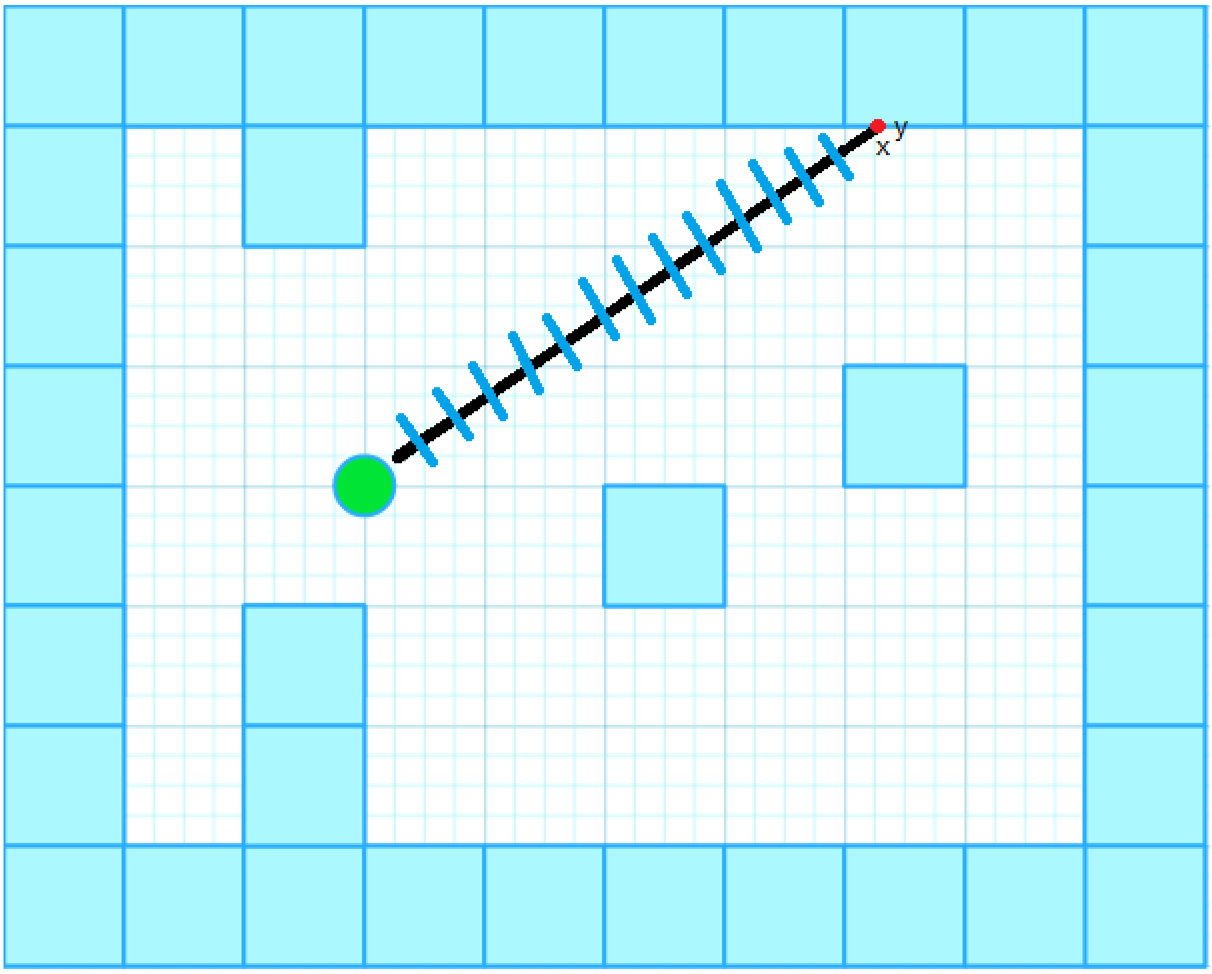
Ray casting is the most basic computer graphics rendering algorithm. In our case ray casting is that we need to cast given number of rays in a certain field of view.



As you can see on the image above, we are casting rays to determine where exactly are the wall tiles compared to our player.

If there is a wall, like in our case the ray stops

What we need to know about the basic technique of ray casting is that we need to define number of steps and the angle that the player is facing, like is shown on the figure bellow:

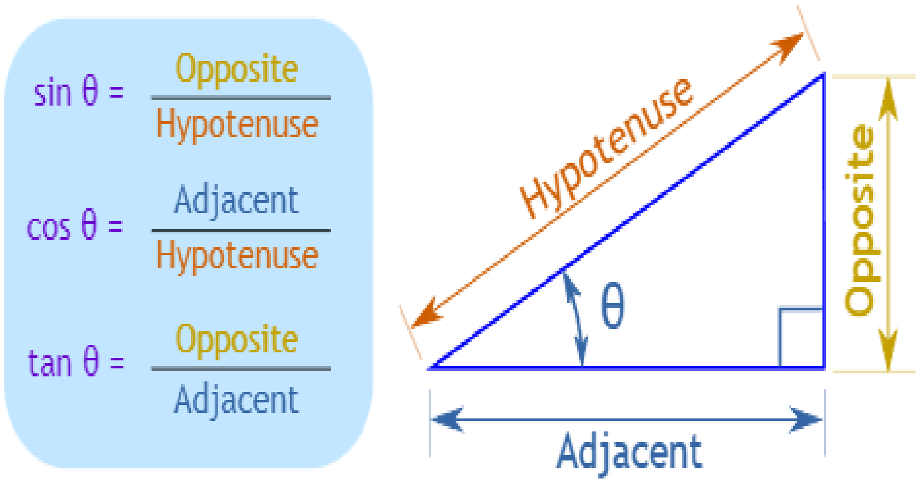


We increase the line with another step if there isn't any object in front of the line, up until we hit a wall. However, as you can imagine to accomplish this, we need to do a lot of computation. This is why we are going take advantage of our tiled map, using the DDA algorithm, but first let me introduce to some simple trigonometric functions.

Basic math concepts

***θ* -** Theta **-** used to represent an unknown measure of angle.

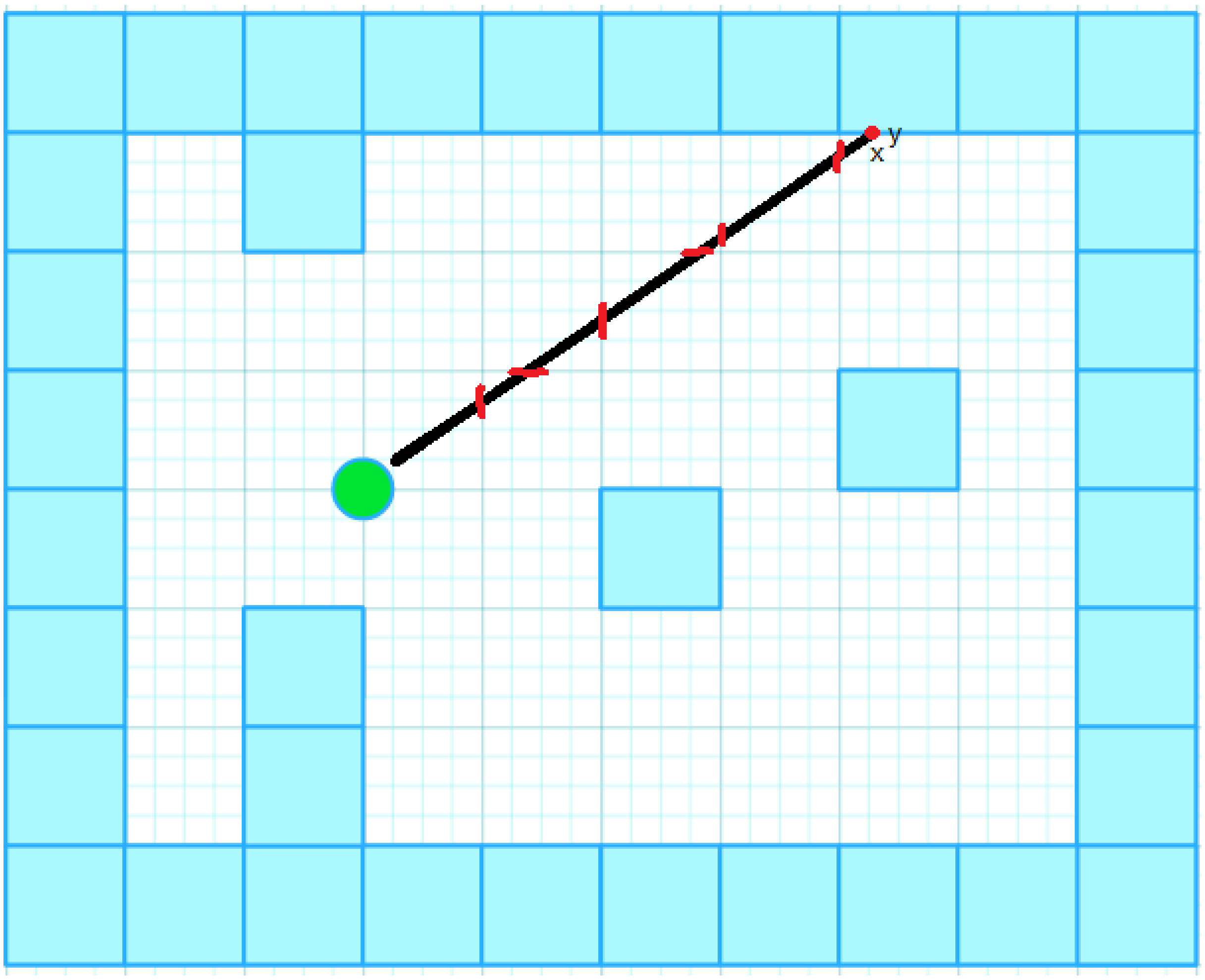
Δ **-** Delta – used to measure intersection of two straight lines.


sin θ = Opposite / Hypotenuse
cos θ = Adjacent / Hypotenuse
tan θ = Opposite / Adjacent
Hypotenuse
Opposite
θ
Adjacent

If we look at the above image, we are going to see the functions sin, cos and tan, which are the main functions in trigonometry. They are ratios of the sides of a right angled triangle.

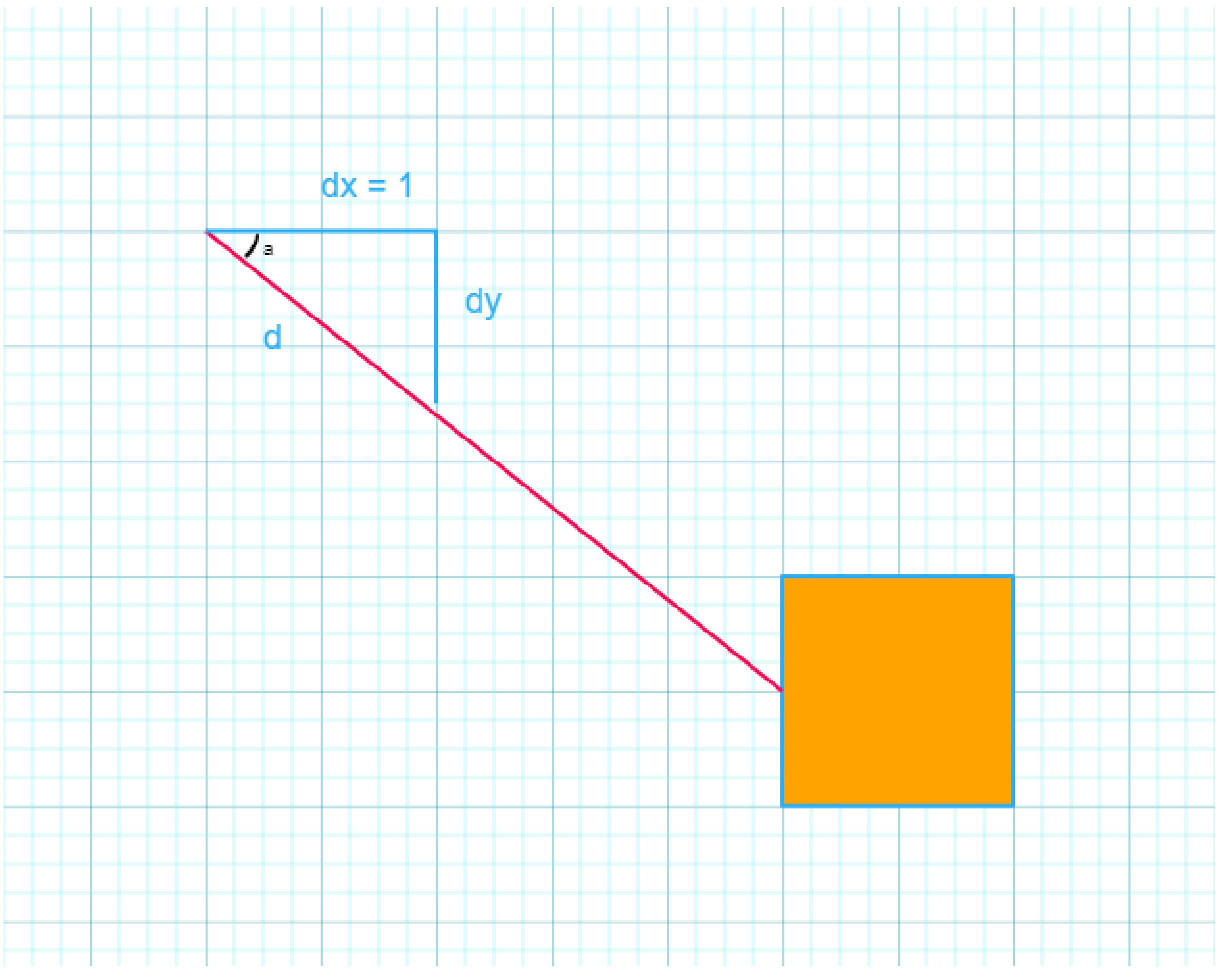
DDA algorithm for fast ray casting

In this implementation of ray casting as we said above we are using the DDA algorithm for fewer math computation. The idea is, because we are on a grid, we can use that grid for the steps of our ray caster:



What basically we are doing here? We are using the tiles on the map to determine the step size of the casted line and looking for horizontal or vertical interception with the tile and here comes the usage of the right triangle.

Lets say that one tile has a size of 1. We know our player angle.



From the above defined functions, we can say that:

$$\cos(a) = dx \;/\; d$$

$$\sin(a) = dy \;/\; d$$

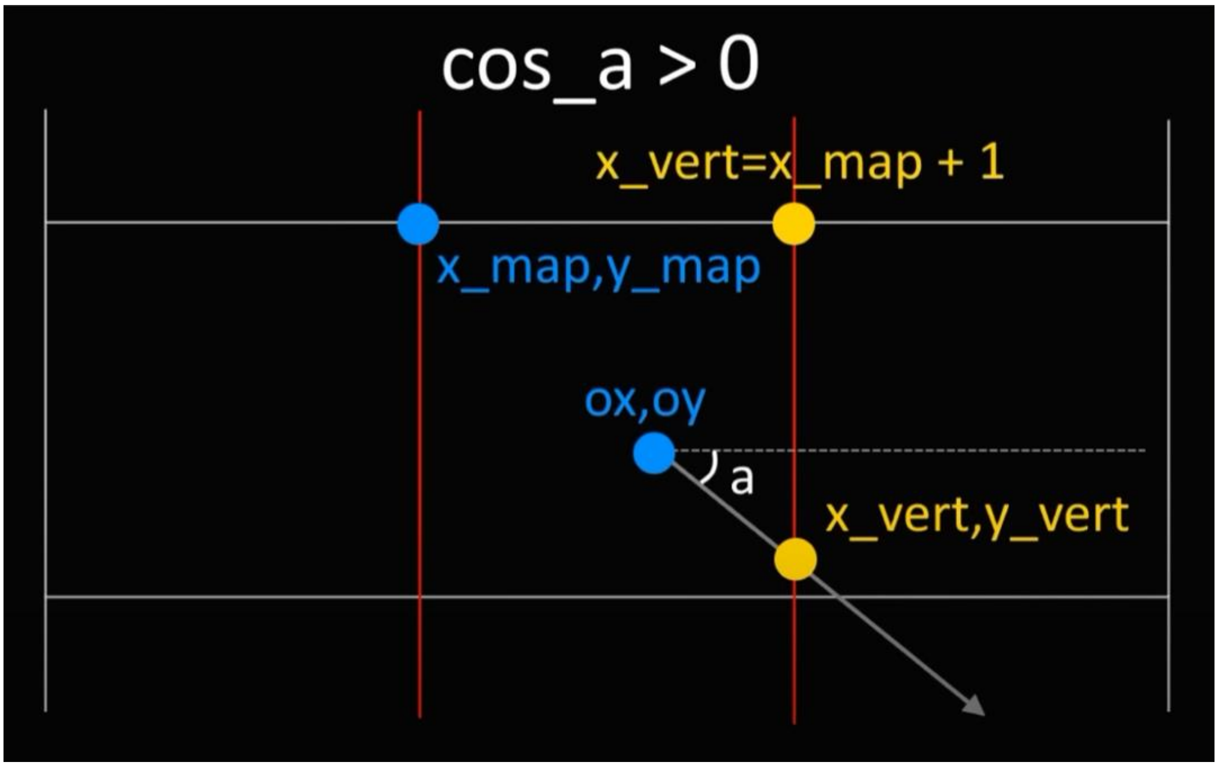
So in order to find the value of d and dy:

$$d = \frac{dx}{\cos(a)}$$

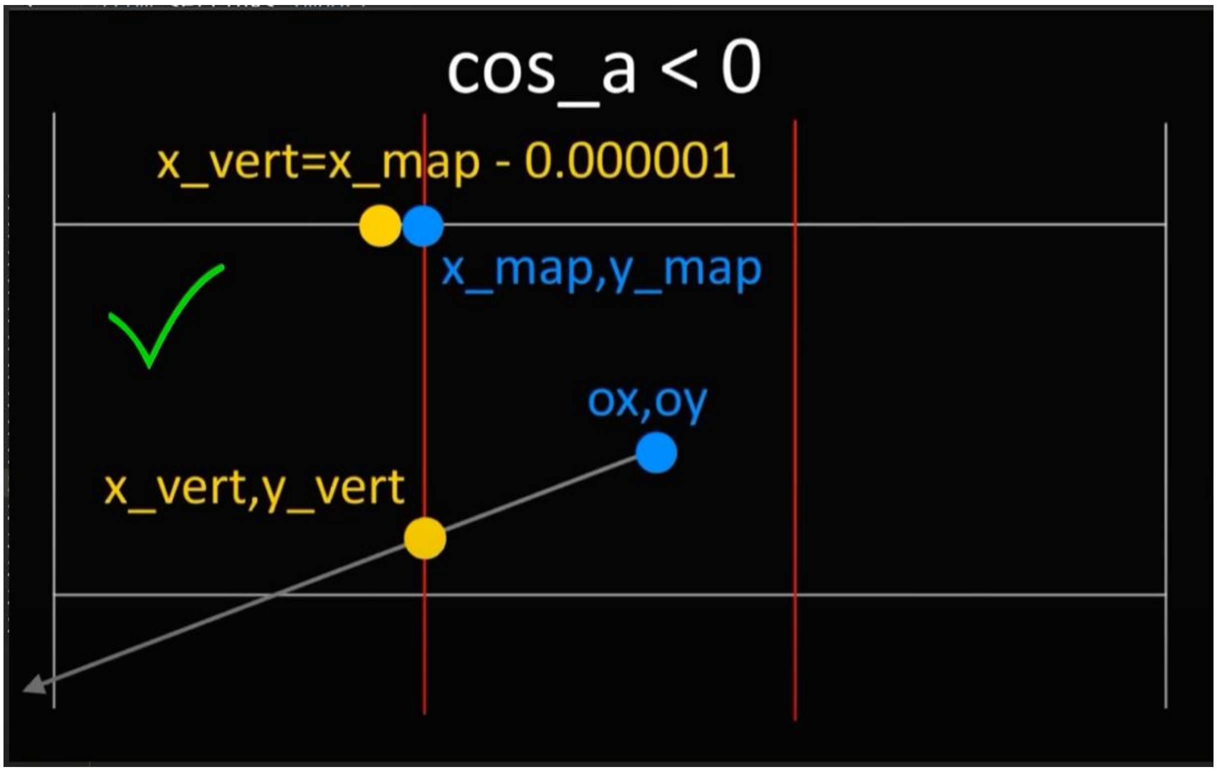
$$dy = \frac{d}{\sin(a)}$$

And, we have found the depth of our ray for one tile. We just need to continue until a wall is hit.

There is a situation where the player position is not in the map tile borders. He is in the tile. In that case we need to calculate the ray up until the first intersection with the vertical tile line or the horizontal tile line and we need to determine the value of the dx or dy for horizontal or vertical interception.



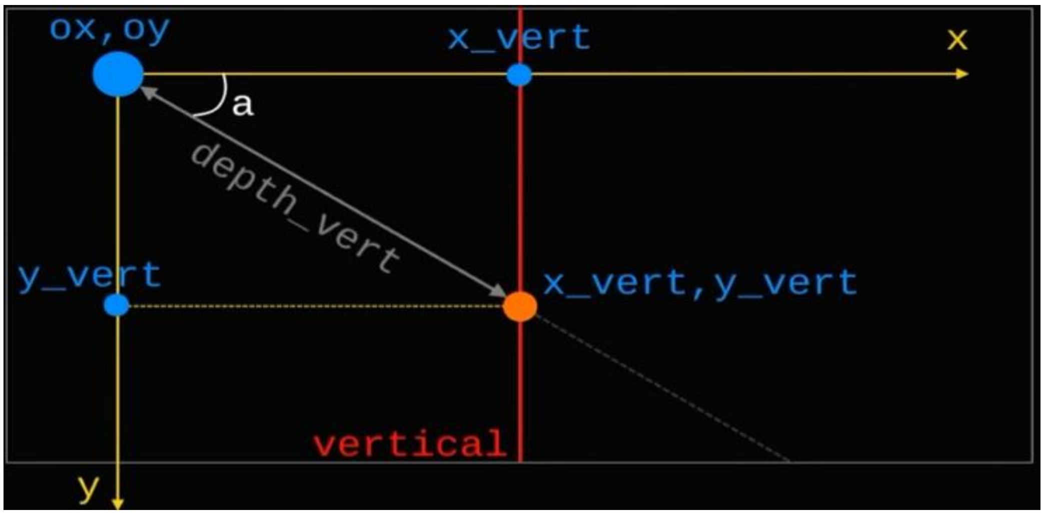
What we need is the tile map position where the player is in, and the player position. So to determine where is our vertical line we just add one to the x_map tile position. If the cos < 0, x_vert is equal to x_map - 0.000001. We subtract small value, because we are looking in the next tile:



`   After we know the value of x_vert, it is pretty straight forward that with the above defined formulas we can determine the depth and the y_vert pretty easy.

depth_vert = (x_vert – ox) / cos(a)

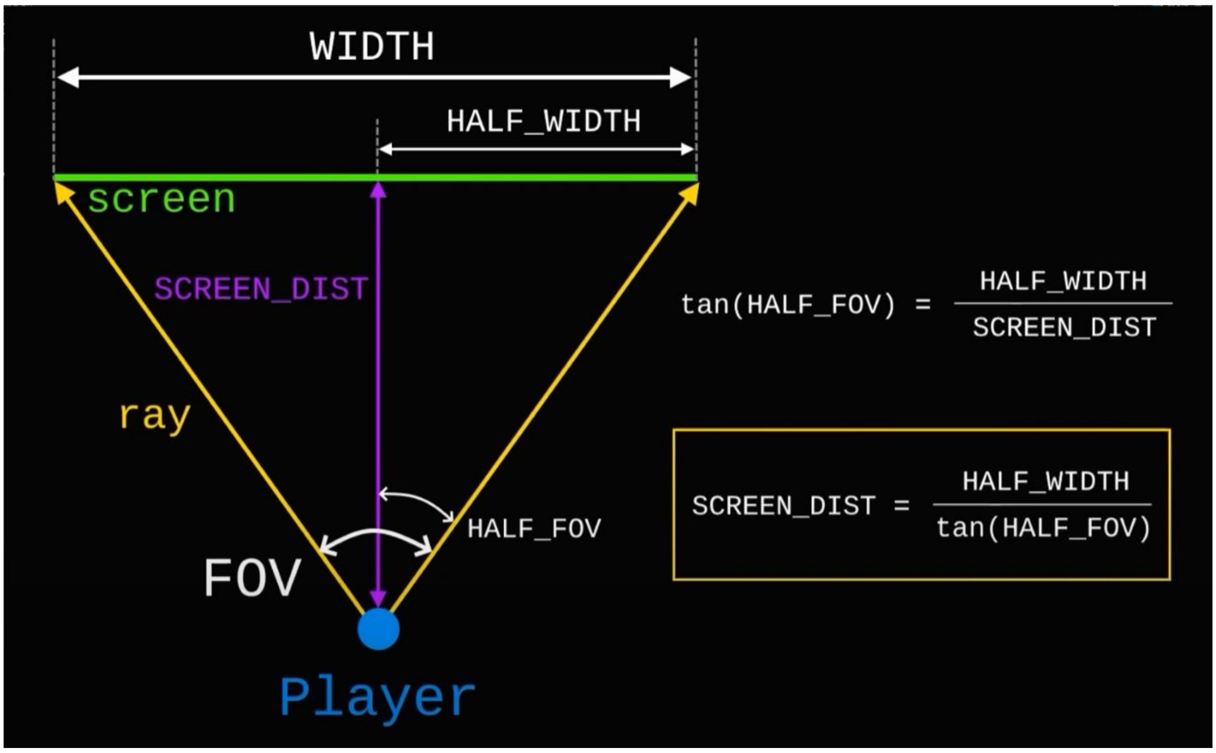
y_vert = oy + depth_vert * sin(a)



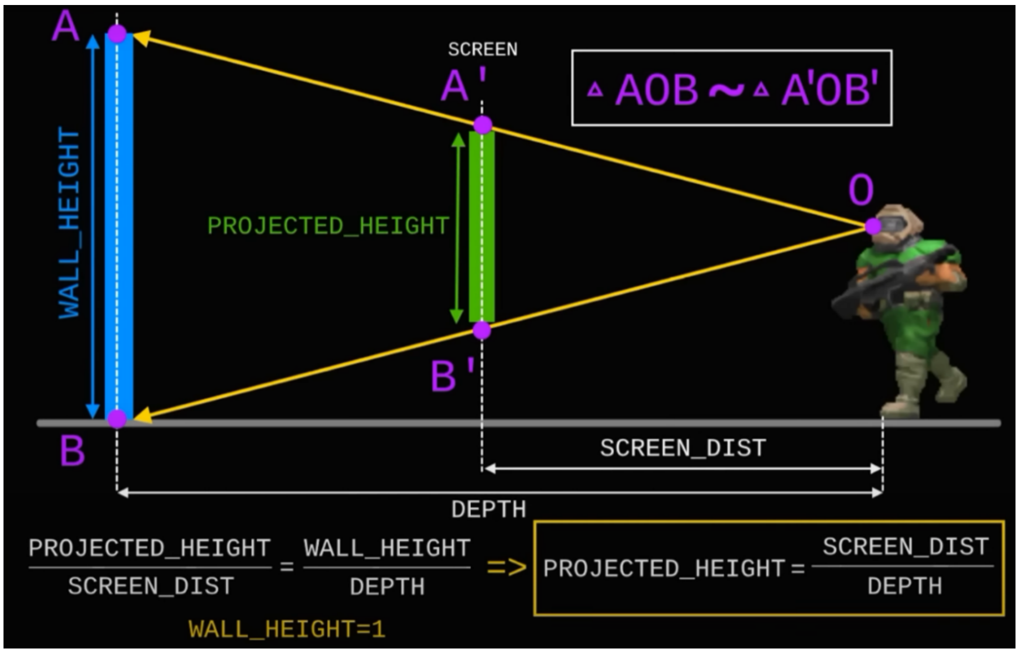
And that’s it. To summarize everything: First we need to get the map tile position, then we need to have the player position and the player angle (the angle that the player is looking at), then we need to check if cos is bigger or smaller than 0 to determine where is the first vertical and horizontal intersection with the tile line. Calculate the depth until our first intersection, and continue calculating the depth and the y_verticals and x_horizontals. We are calculating the ray both for verticals and horizontals and then taking the smaller value.

Texture render

In order to get the projection, we need to place a screen at specified position from the player:

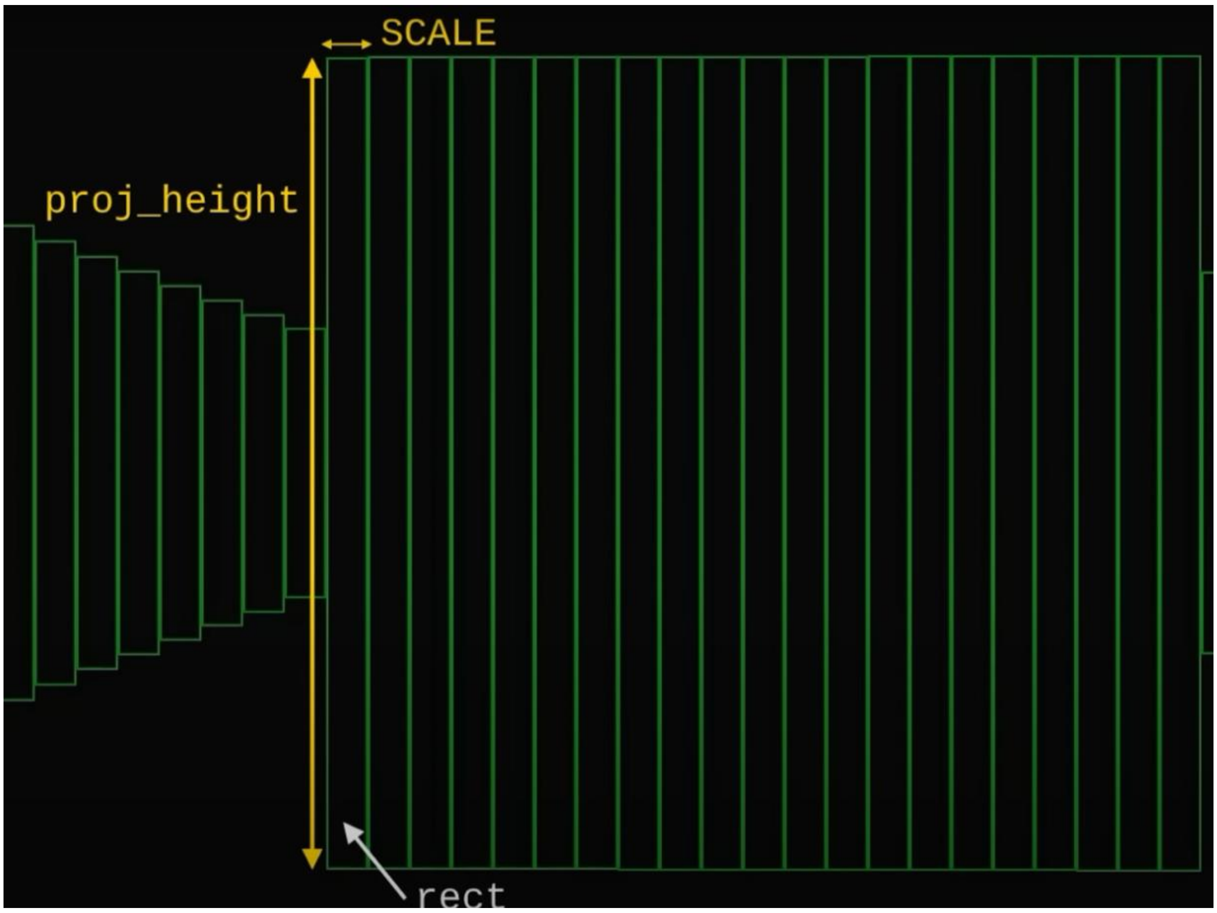

WIDTH
HALF_WIDTH
screen
SCREEN_DIST
ray
HALF_FOV
FOV
Player
tan(HALF_FOV) = HALF_WIDTH / SCREEN_DIST
SCREEN_DIST = HALF_WIDTH / tan(HALF_FOV)

Here is how to calculate the projection height:


A
SCREEN
A'
△AOB ~ △A'OB'
O
WALL_HEIGHT
PROJECTED_HEIGHT
B'
B
SCREEN_DIST
DEPTH
PROJECTED_HEIGHT / SCREEN_DIST = WALL_HEIGHT / DEPTH => PROJECTED_HEIGHT = SCREEN_DIST / DEPTH
WALL_HEIGHT=1

However, we add a small number to the depth to avoid division by 0 error.

In order to show textures on the ray cast, need to divide the texture in smaller rectangles:



One important thing to notice here is that we need to calculate the offset of the textures. We will consider all four cases for that when cos < 0 and cos >0.

If cos > 0

Offset = y_vert %1 – where 1 is the wall size

If cos < 0

Offset = 1 – y_vert % 1

Same applies for horizontals.

Sprite projection and getting the sprite

The first thing we need to do is to determine the that angle. We use the atan2 from the math module and input the player and the sprite positions. Next, we need to find the delta which is the difference between the player angle and the theta angle. Next, we get the delta rays by dividing the delta by delta angle and we find the x position on the sprite on the screen by adding the half number of rays with delta rays and multiplying by the scale.

For calculating the sprite size, we need to calculate the distance to the sprite.

To get the projection of the sprite we need to calculate the height of the sprite projection adjusting by the image ratio.

After that we are finding the sprite position on the screen and add it to the list of walls to get rendered

BSF

- Declare a queue and insert the starting vertex.
- Initialize a visited array and mark the starting vertex as visited.
- Follow the below process till the queue becomes empty:
- Remove the first vertex of the queue.
- Mark that vertex as visited.
- Insert all the unvisited neighbours of the vertex into the queue.

Resources

https://killerrobotics.me/2021/09/26/developing-a-raycasting-3d-engine-game-in-python-and-pygame-part-5/

https://www.spriters-resource.com/pc_computer/doomdoomii/

https://trinket.io/python/acfadf20d9

https://fontmeme.com/doom-font/

https://www.youtube.com/watch?v=4gqPv7A_YRY&t=554s&ab_channel=FinFET

https://lodev.org/cgtutor/raycasting.html