2D Raycasting

Lukas Jonca
2020-05-04
ICS4U

Introduction:
Ray Casting is an algorithm that is slightly outdated and allows for simplistic 3D rendering and other applications such as simple game AI's, in this project the focus will be to create a 2D representation of the raycast algorithm. The applications for the project later could be vast and the code could be reimplemented into other projects.

Functional Requirements:
For the project to be functional it must be able to complete the following.

- Have a UI that the user can use to understand what is happening.
- Take valid user input and deal with invalid input accordingly.
- Update and keep track of the GUI screen.
- Keep track of rays directions and coordinates.
- Keep track of walls and coordinates.
- Calculate intersections of rays and walls.
- Calculate distances between walls and rays.
- Save and load files.

For the program to work it is vital that all calculations are done properly and give correct answers for line intersection.

The program will serve as a demo and will be designed to be easily implementable in future projects. First a simple program will be made for determining line segment intersection this will allow for easier testing of the equations. After the functions will be implemented in the raycaster to determine where the rays collide with the walls. The GUI will visualize the l rays and walls cutting of the rays where they coloide.

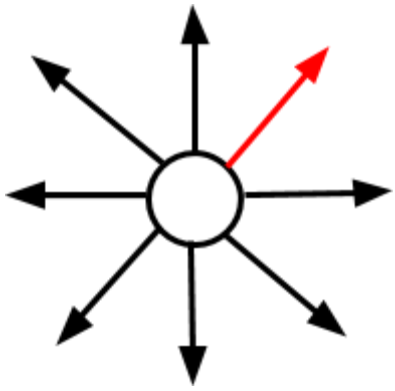
Non Functional Requirements:
To enhance the users experience, requirements are as follows:

- An attractive and functional GUI.
- Error and case handling.
- Proper well formatted documentation
- Compelling graphics.

To provide a clear representation is going on the graphics must cleanly display the rays intersecting with the walls and stopping. Error handling will also be a main priority, humans making mistakes when inputting information is inevitable so it's important the program is able to handle these improper inputs.

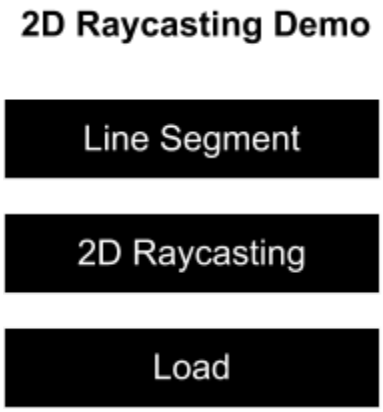
Core Features:

Main Menu - will display various options for running different parts of the program.
2D Raycast GUI - will display a 2D visualization of the rays interacting with walls.
User Controls - the user will need to be able to interact with the program and control the character.



The player will have a main direction that they are facing which will be used to change the speed X and speed Y.

User Interface:



Main Menu: The main menu will be a simple and clean method of navigating the different parts of the program.

Line Segments: Will be a simplistic visualization of line segment intersection without much user input.

2D raycasting: Will allow for complete control of player movement and will visualize the rays hitting the walls.

Calculation and Functions:

For the project to be functional calculations will need to be made using the following formulas,

Distance:

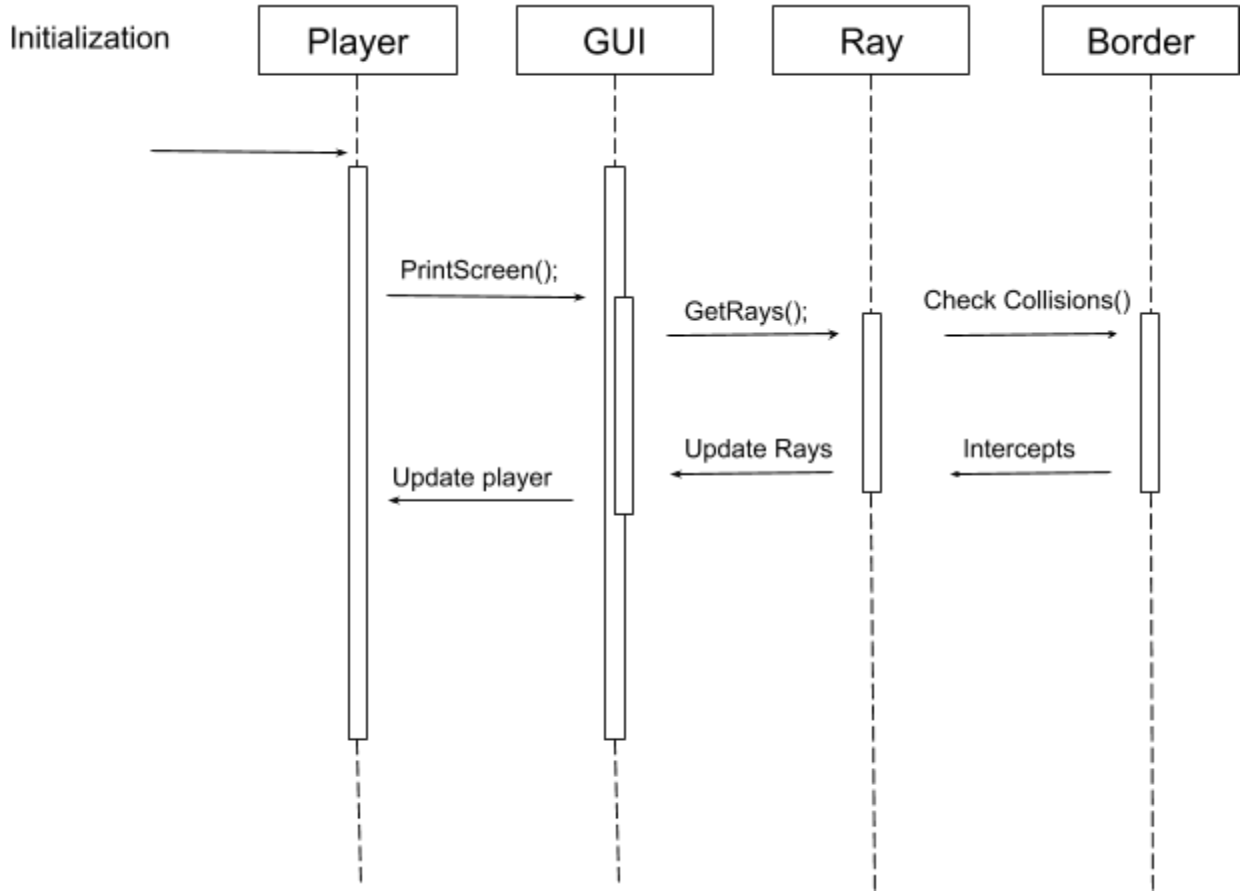
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Will be used to calculate distances between points.

Line Segment Intersection:

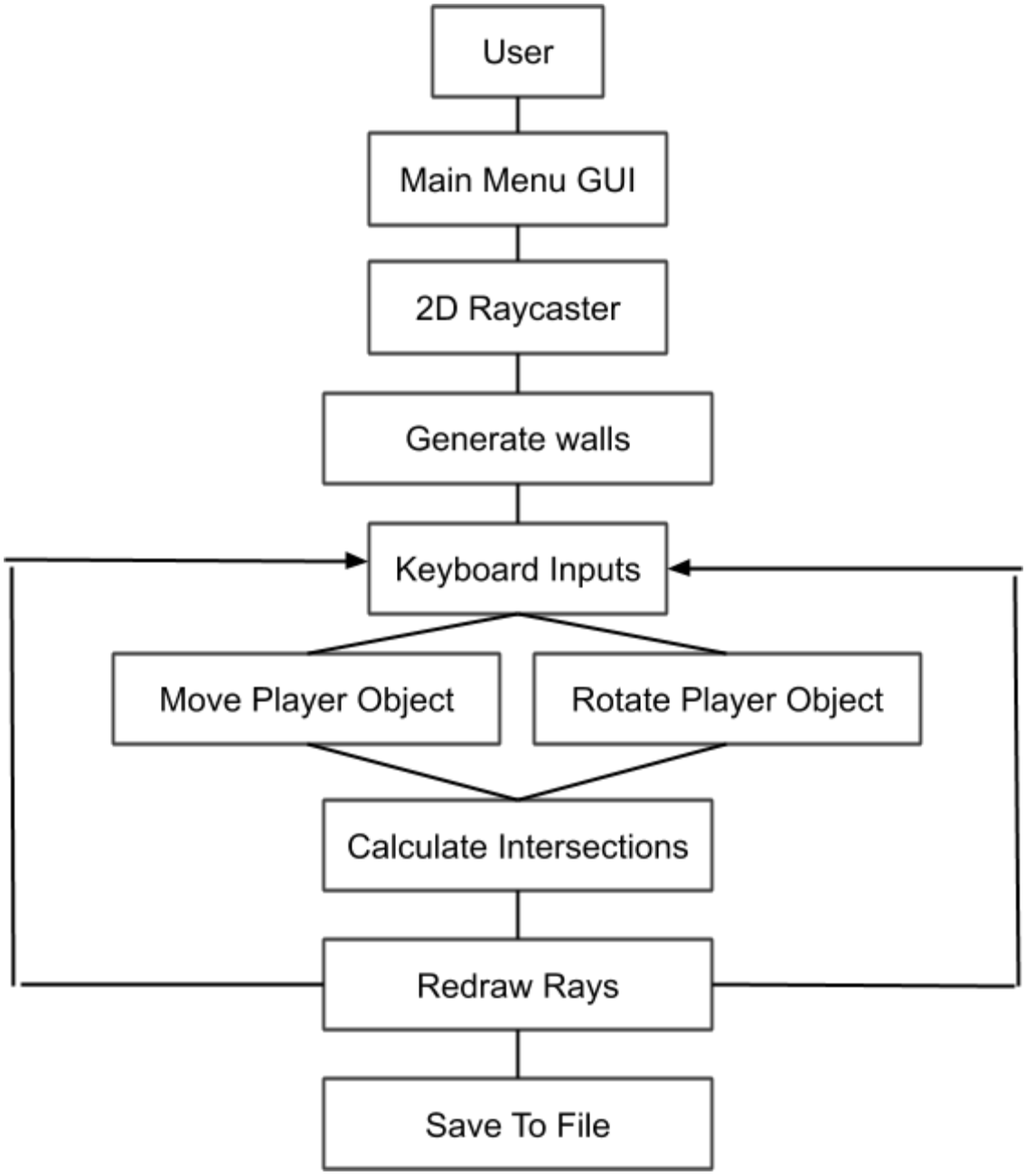
$$(P_x, P_y) = (x_1 + t(x_2 - x_1),\ y_1 + t(y_2 - y_1))$$

Will be used to calculate where line segments intersect with one another, this will need to be done many times throughout the program so its important it works properly for all cases.



Sequence Diagram:

- Players objects will be initiated in the Raycast GUI.
- The GUI will take the player object and display it using the variable housed within, allowing for easy access between different parts of the program.
- The rays will then be checked for any collisions with the wall and adjusted accordingly.
- Then the GUI and player will be updated with updated rays.
- The player rays will need to be rechecked after each movement and updated.



Use Case Diagram (Updated):

- The demo will be launched using the main menu and then walls will be randomly generated on the screen.
- The user will be able to interact with the demo through the keyboard allowing for the player object to be moved around the screen.
- After the player coordinates are updated the rays will need to all be rechecked for collisions.
- Finally if the user would like to save the player's position and map layout they can save the playe object for later.

Milestone:

- Calculating simple 2 line line segment intersection and visualising.
- Creating a player from where rays radiate in all directions from.
- Generating random walls on the screen.
- Integrating player movement.
- Calculating wall and ray intersection and redrawing the screen.

Proposal:

Although raycasting is an older algorithm it is still a good place to start as it still may allow for the exploration of 3D rendering later on and has applications for AI movement in games. The program needs to be built in a modular way where the code can later be easily implemented. The goal of this project is to create a 2D visual representation of 2D ray casting.

Glossary:

Graphical User Interface (GUI) - how the user interacts with the actual program graphically.
Array List - a collection of data in a list of stored data.
Raycasting - the use of intersections to render simple perspective images.