Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

**Department of Informatics**

# CONSOLE GAME:ASTROWORLD

**C++ Console Project**

**Name: Gurdeep Singh**

**Neptun Code: ERXIPV**

**2020.**

## Table of Contents

1. Introduction/Program Description ........ 4
*1.1 Solution Design* ........ 4
*1.2 Solution Algorithm* ........ 6
*1.3 External Solution* ........ 9
*1.4 Changes from Original form* ........ 9
*1.5 Testing* ........ 9

# Table of Figures

*Figure 1: Initial idea of Game functions and Classes.................................................................................... 4*
*Figure 2: Hitbox () function idea at initial stage for destorying enemy and damage taken by hitting stone. ........................................................................................................................................................ 5*
*Figure 3: HP bar reduction and gain idea including stone and powerup objects......................................... 5*
*Figure 4: graphic representation of how velocity works in-game............................................................... 7*
*Figure 5: making of ship class and hp bar.................................................................................................. 10*
*Figure 6: creation of attackship class with collision testing........................................................................ 11*
*Figure 7: making of stones class while testing collision without losing the stone square and decrease of HP via Hitpoint() function. .............................................................................................................. 11*

# 1. Introduction / Program Description

Astroworld is a C++ console game with light use of graphics. It is game that is been inspired from 1984 “Astropanic” developed by “Compute! Publication”. I tried to capture the nostalgia feel of old game with a little change in features from the original game. “Visual Studio” is the IDE that has been used to make this project and a light chilli framework for the creation of the graphics.

*Source: Astroworld inspiration* - https://www.youtube.com/watch?v=-GYySwKia-Y

## 1.1 Solution Design

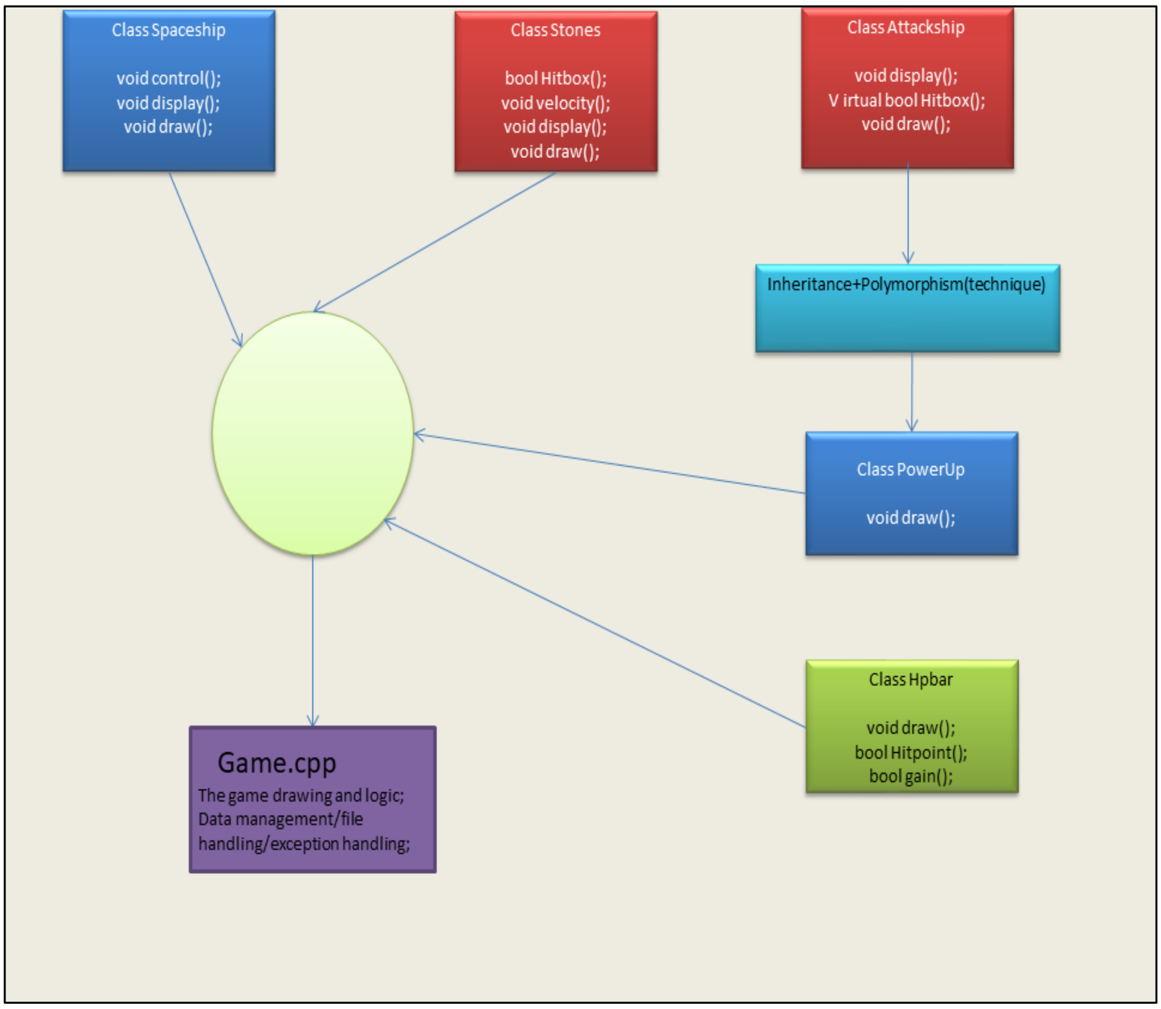

**Figure 1: Initial idea of Game functions and Classes**

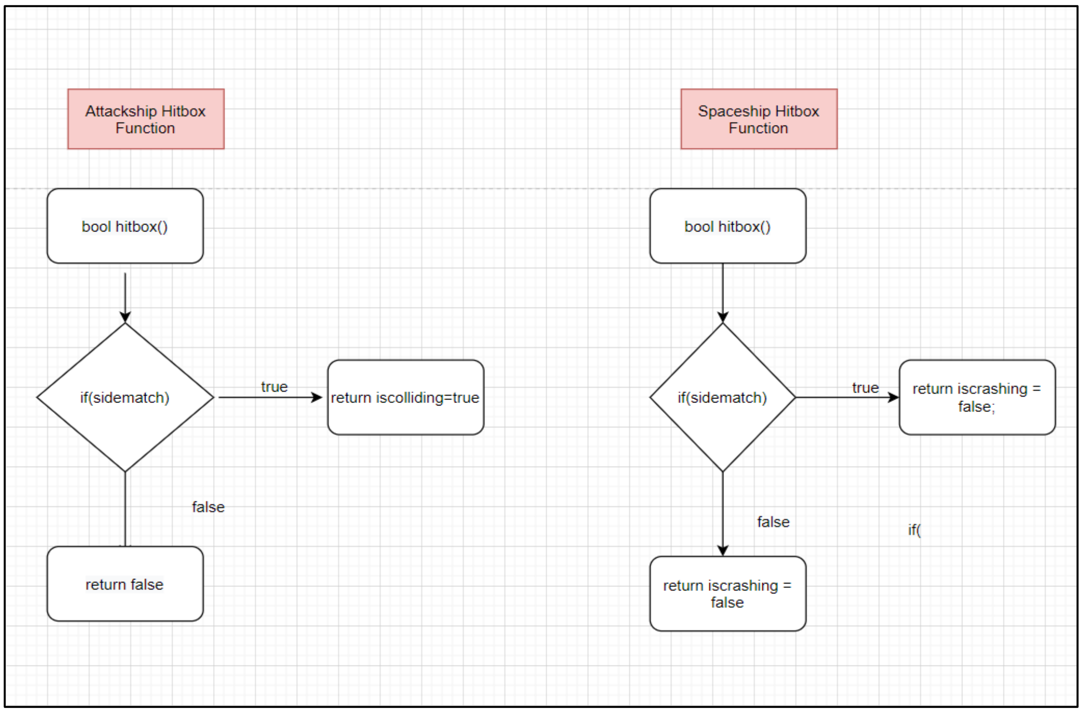

**Figure 2: Hitbox () function idea at initial stage for destorying enemy and damage taken by hitting stone.**

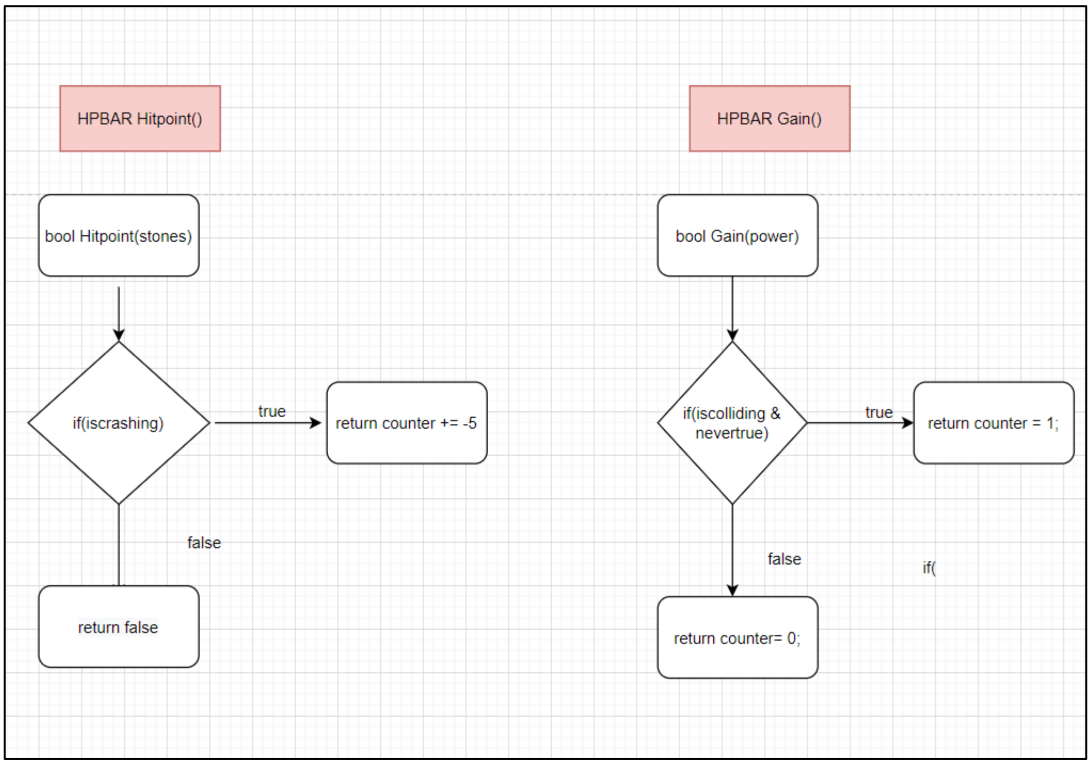

**Figure 3: HP bar reduction and gain idea including stone and powerup objects.**

## *1.2 Solution Algorithm*

As shown in solution design, the first thought at the start of project was to make different classes representing different things (objects) in the game and the kind of function that were going to be inside those classes. These Classes were:

**Class Spaceship** ( void display, void control, void draw )
**Class HPbar** ( bool Hitpoint, void Gain, void draw)
**Class Attackship** (void Velocity, virtual bool Hitbox, void display, virtual void draw)
**Class Stones** (aestroids) (void Hitbox, void display, void draw)
**Class PowerUp**( void draw and bool hitbox)

All the above classes were intertwined with the game class where all of them come together to make this game. The different technique that has been used in the project are the object oriented paradigm i.e (Classes, Data Hiding, Encapsulation), File handling, Data Management, Exception Handling Inheritance, Polymorphism.

**Object Oriented Paradigm:** The classes have been mentioned before. All the variables are private and only necessary information is made public. There are getters available for all the data variables in each data classes.

**File handling & Data Management:** Both of these techniques are used in the main (game.cpp) file. The main purpose of file handling technique was to save scores into the file while the data management is used to dynamically write the data available in the original file to a backup file in case the original file gets corrupted.

**Exception Handling**: It was used to safe check the HP bar in case it goes below 0.

**Inheritance:** This technique is used in the powerup, which inherits all the properties from the class Attackship.

***Polymorphism***: Two function inside the the powerup class void draw and bool hitbox has some changes compared from its parent class attackship so, attackship's void draw and bool hitbox has been made virtual.
void draw inside powerup makes a med kit sprite model while void draw in the attackship make alien ship sprite model.
bool Hitbox in powerup is just for collision detection while the bool hitbox in the attackship is to update the countscore() function (explanation below) and for collision detection.

***Algorithms: <u>Score System</u>:***
***Start variable***- Uses chrono libraray to calculate current time.
***<u>void elapsed() & Countscore():</u>*** "The countScore() method is used to increment the player's score when a new spaceship is captured. The added score for one spaceship starts at 50 points and gets smaller as time passes, meaning the player should capture all spaceships as fast as possible. In order to make the score smaller as time passes the elapsed() method is used. The elapsed() method uses the std::chrono standard library

to get the current time and subtract the time at which a spaceship was last captured (variable 'start'). 'start' is set to the current time by default when the game launches and gets updated every time a new spaceship is captured. After getting the total elapsed milliseconds (using std::chrono's 'duration_cast') from the last spaceship capture we divide the base score (50) with those elapsed milliseconds and add the result to the player's total score."

***Collision Detection (Hitbox):*** This is the algorithm that helps us to find when the two objects are colliding. It's basically present fully or partially present in all the classes one way or another. The main object here is the main spaceship that is being controlled by the user. The Hitbox a term used in the modern gaming industry is a function that calculate all the side of main object ship with the other objects size from top and bottom to left and right using greater than, smaller than and equal to operator. If any of the sides are even slightly touching it'll be consider as colliding.

***Velocity Concept & void Display:*** The velocity concept for self moving objects is really simple. Objects constructed at the random places (using rand function) starts moving at different direction by updating there velocity variable in x and y randomly using same rand function.

The idea of this algorithm came from the void Control function() used in the main (user controlled ship). Using the framework commands When certain keys are pressed i.e (if(wnd.kbd.keyisPressed(VK_UP)) the object that has bind to this function starts moving in the up direction. It starts moving up because we are adding certain pixel in that direction. E.g x = 400 and I add 2 to it then x will be moving by 2 in x- axis. So if it got subtracted by -2 it will move two pixel right.

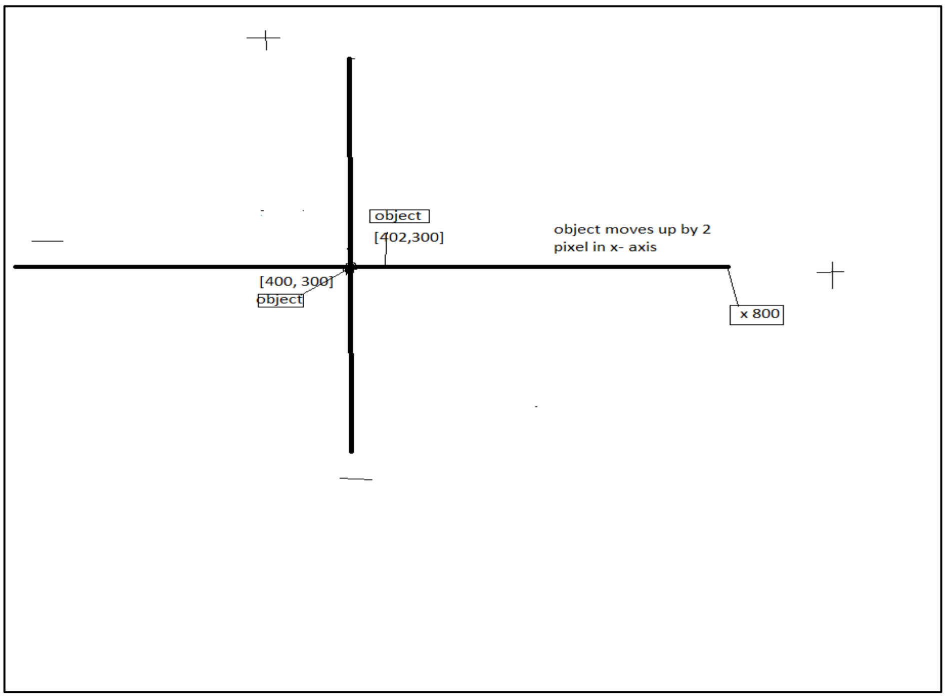

**Figure 4: graphic representation of how velocity works in-game**

But what happens if it reaches at the end of screen like 800 pixel on x axis. Now, here comes the Display function() in play if object goes below 800 or equal to 800 it will get back pushed back to the -1 rightful pixel w.r.t its height and width, same goes for y axis . So, it will not go beyond the 800x600pixel. Therefore, we can apply our velocity concept here.
To move the object in any certain direction we just need to add 1 its current location without any virtual key binding it. It will start accelerating to that direction as it hits our display wall we know it was accelerating toward certain direction so all we need to do is decelerate it to the other direction. i.e x = 400, velocity = 1,
Then dir = x+velocity; ( starts moving toward right)
When it hits the x=800, it get pushed back in the display w.rt height or width. And we start decelerate in opposite direction.
dir= -dir;

***<u>HP Bar Drawing + Hit and gain point function</u>***: Algorithm uses basic shape design for loop idea. But the prominent thing here is the counter which is interwined with different objects like stones and the powerup which are passed by the reference in order to get the colliding and crashing bool into the hitpoint and gain function. So in case it is colliding with the power the HP bar will be refilled(redraw) while if its crashing with the stones(asteroid)objects it will decrease it with -5 rate. In case it goes to zero gameover screen will show up in screen.

***<u>Storing the score and backup file Algorithm:</u>*** The algorithm uses fstream library for file handling. There is a ofstream file called "writeHandler" and ifstream file called "readHandler". First the file is opened in the write mode with append so in case there is no file a file is made using append. We write the Timestamp (using timestamp function) of when the user finished the game and the time score is getting added in the file. Roundf() is used to round off the score. After this we close the file.

This was score storing algorithm. Now, to make a backup file we open file in read mode using readHandler, using istreambuf_iterator to calculate all the characters in succession till the end line "\n" . Getline() returns true if there are still lines in the file. It takes an ifstream variable (read mode (std::ios::in)) and the string where to put the current line. All the lines get saved inside the line variable. Then the current line ('line' variable) gets saved to the dynamic array. Index will keep getting increement and it'll go to next line till there are no lines. At last we will open write handler and write all the lines inside the backup file.

***<u>Sprite Drawing:</u>*** At the end, Spirit modeling was done by in-built class surface and graphics which uses bunch of other functions from framework to make a the model. In case of single object, the surface class was initialized inside the class e.g spaceship while in case of multiple object it was initialized in game.h with the directory location of the image. Only image type that is allowed is bgm. So, the whole program doesn't waste large memory I passed the surface as a pointer, so all the objects uses same surface to draw sprites.

## 1.3 External Solution

The following is the list of function in framework that has been used to make graphics in my project:
Class Graphics (putpixel and drawsprite function are declared inside it.)
Class Surface, Class Recl, Class Vei2, Vec2, Colors (makes surface in console screen to draw sprite model in it, different colors in bmp image and for other model related things)
Class Frametimer (To calculate the fps)
Class Keyboard, Mouse, Resources,chilliwin,DXErr, chilliexception, Mainwindow (These all classes are window api function used to create this framework)

Beside that the timestamp function has been taken from stackoverflow.

```
// get current local time
auto t = std::time(nullptr);
auto tm = *std::localtime (&t);

// format the current local time to a string
std::ostringstream oss;
oss << std::put_time(&tm, "%d/%m/%Y %H:%M:%S");

// assign to our name variable
name = oss.str();

this is the code
```

## 1.4 Changes from Original form

The whole project was mostly the same as I intended in the form, the only changes that took place in the project were the change in the way techniques got used.

***Inheritance:*** It was used to make Class powerup from Class attackship while in form I stated making enemy ship from base ship class but their functioning is different.
***File Handling:*** It was used to store score and make a backup of scorelist instead of loading to make pure pixelated sprites, it would have been not very optimal .
***Multiple Inheritance:*** I already stated in my form that I might use this. But I ended up choosing not to.
***Dynamic Memory Management:*** After making HP bar from for loop design, I was very happy with the way it was functioning.

## 1.5 Testing

The first class completed during the project timeline was Class Spaceship. Class spaceship functions are already well explained in the chapter 1.2. At first there were no models of spaceship so I just draw a square
Loop design 1: for square

```
    for (int i = b; i < b + height; i++)
    {
    for (j = a; j < a + width; j++)
    {
    gfx.PutPixel(j, i, 0, 153, 0);
                         }
    }
```

HP bar model's getting refilled and decreasing  depends upon the bool condition of Class stone (asteroid) and powerup(child class of attackship).

```
    if (stone.iscrashing_() )
    {
           return counter += -5;
    }

    if (power.iscolliding && !power.neverfalse)

    {
    return counter = 1;
    }
```

Here, we witness the creation of counter.
Counter is always updating when colliding and crashing. Thus the change in the model of HP bar took place. Where we add counter's value to width of the hp bar in the same for loop design, just with the addition of counter in the second for loop with width.
Loop Design 2: for HP bar:

```
    for (int i = b; i < b + height; i++)
    {
    for (j = a; j < a + width + counter; j++)
    {
    gfx.PutPixel(j, i, 0, 153, 0);
                         }
}
```

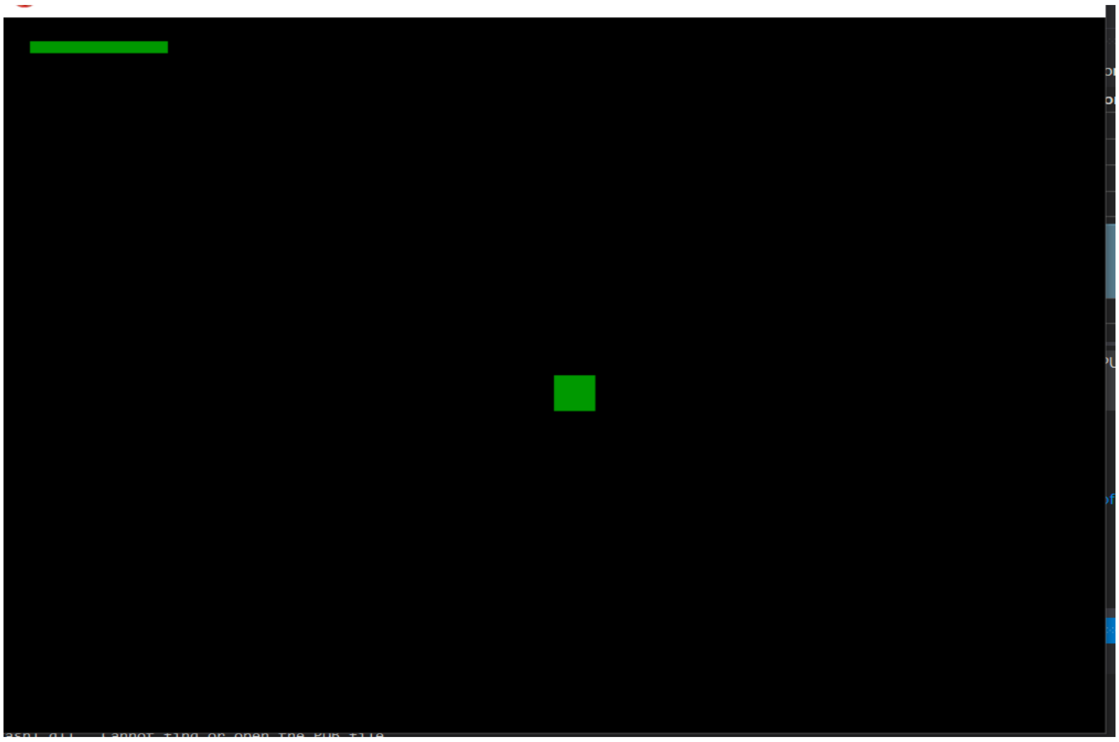

**Figure 5: making of ship class and hp bar**

After that I made sure all the function of boom(ship) works. The same square model for the Class attackship (enemy) was made. And the function described before were made. The ship boom was passed as reference into the hitbox function of attackship and after collision detection they made bool iscolliding() true which was initially set at false which resulted in ship destroying red marker.

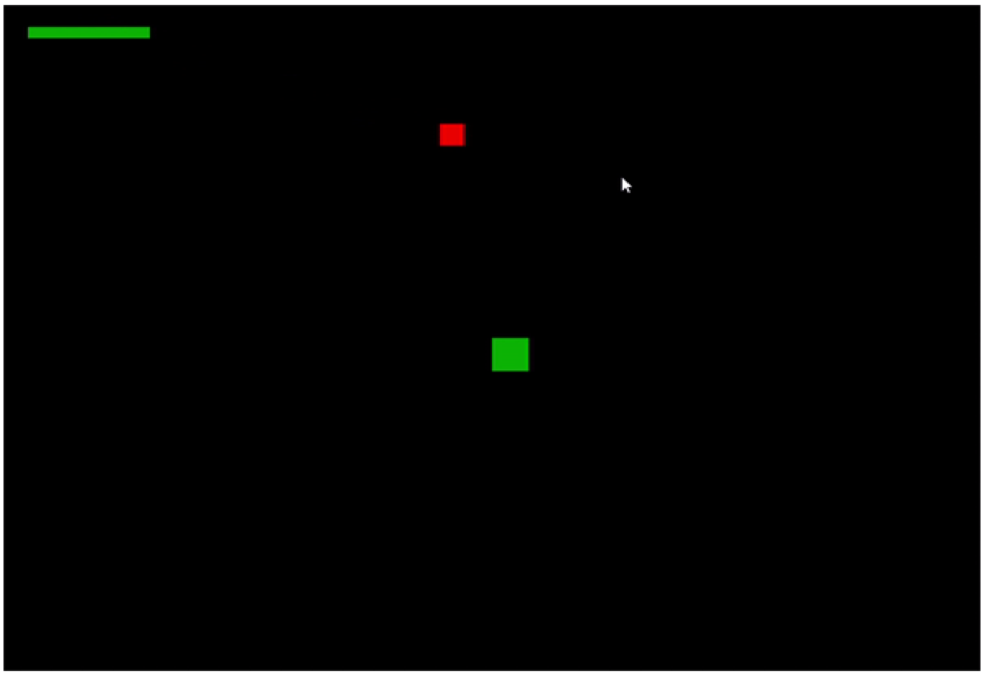
**Figure 6: creation of attackship class with collision testing**

*(link to the gif with the model working)*
[https://im4.ezgif.com/tmp/ezgif-4-f489e6216c17.gif](https://im4.ezgif.com/tmp/ezgif-4-f489e6216c17.gif)

After testing attackship, class stones was made as same square model and when they are colliding (ship and stone) a bool function iscrashing will become true but as soon as they go far away from each other (iscrashing will again become false ) which will result in the square not getting destroyed and staying in shape. As I mentioned the HP bar will decrease since green square object is crashing with the white square one. (more detailed in the project comments).

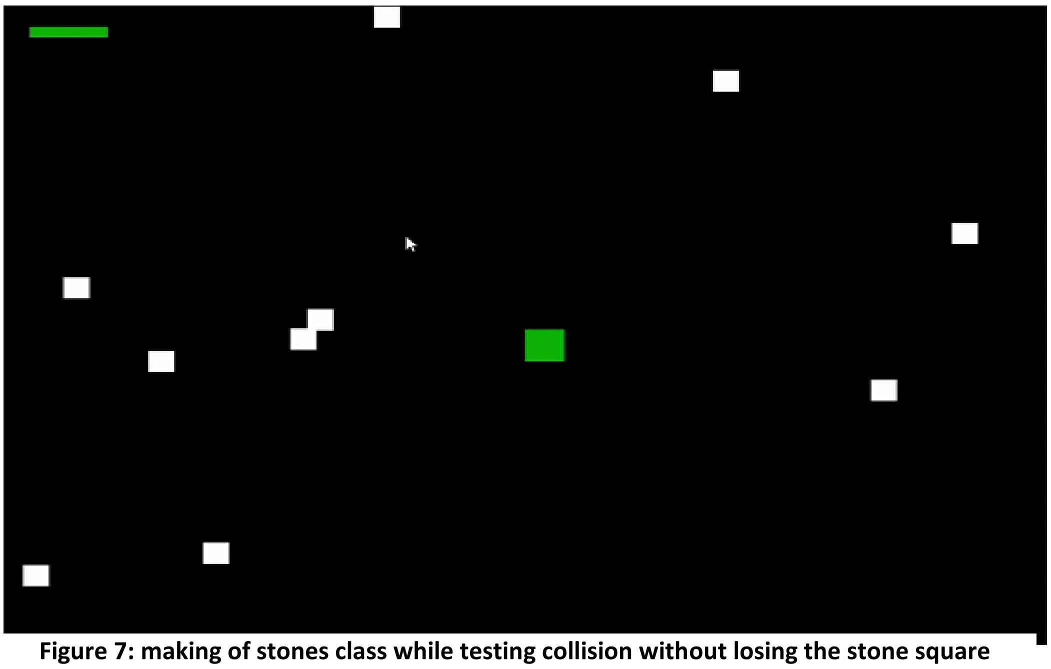
**Figure 7: making of stones class while testing collision without losing the stone square and decrease of HP via Hitpoint() function.**

Source for gif: https://im4.ezgif.com/tmp/ezgif-4-ebcc95f03ba9.gif