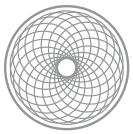


David Haro

BattleShip

Version: 2.5.1

Project 2

CSC 17A

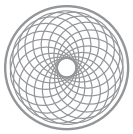
Prepared for: Dr. Mark Lehr
Prepared by: David Haro

December 8th, 2014

Attachments

Doxygen files

BattleShip V2_5_1 netbeans project folder



David Haro

Summary

Objective

Successfully plan and design a game in form of a C++ program that exemplifies the learning of this course. Methods and strategies that are desired are those that have been recently reviewed. This includes, but not limited to: Single and multidimensional arrays, dynamic memory allocation in 2D arrays, passing dynamic 2D arrays to functions, clearing memory when complete, and use of structs/

Goals

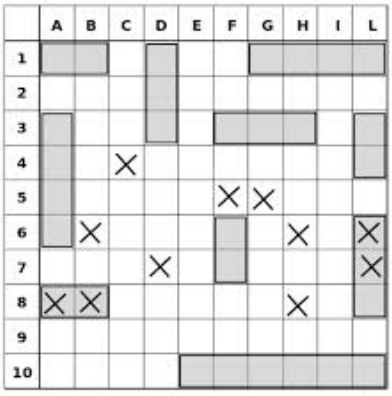
Develop and improve a previous version of Battleship/Salvo Game by using C++ knowledge. Fine tune methods that were previously covered during lecture, lab, and homework assignments.

The Game

The Game puts the player against the Computer in a game of Battleship. This version of the game is based off of the paper version once called Salvo. First the user must read the rules or directions is unfamiliar with game, then proceed with choosing coordinates where ships are to be deployed.

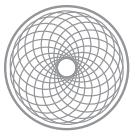
The Coordinates are put through series of validations in order to determine that they do not fall out of bounds or criss cross over other ships. Once the CPU selects its positions, the players are then asked to pick a strike point, and this is a mere hit or miss function.

The game ends when one of the players is out of ships on the board. The Program ends with displaying the winner.



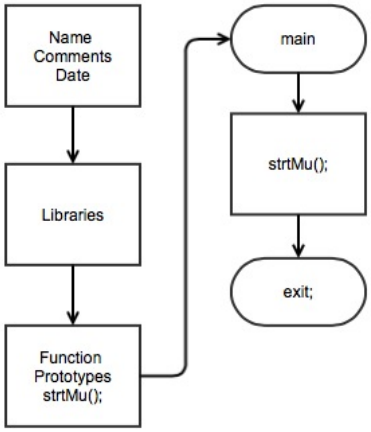
Project Size

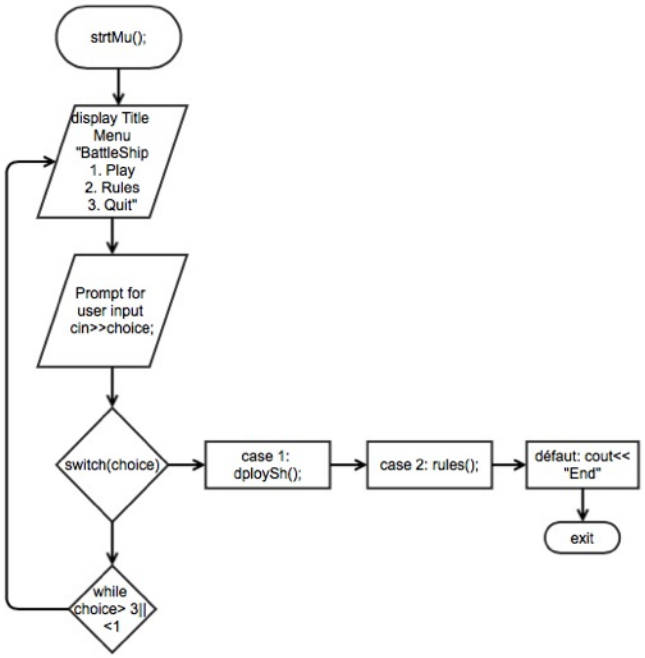
- Roughly 1,100 lines of code, this include about 80 lines of comments
- Number of 2D Arrays: 3
- Dynamically allocated arrays: 2
- Methods Used: Functions, returning Primitive data types, passing functions, single arrays, linear searching of 2D Arrays, structs, dynamic memory allocation etc.
- Classes, inheritance, reading binary files, polymorphic behavior are included.

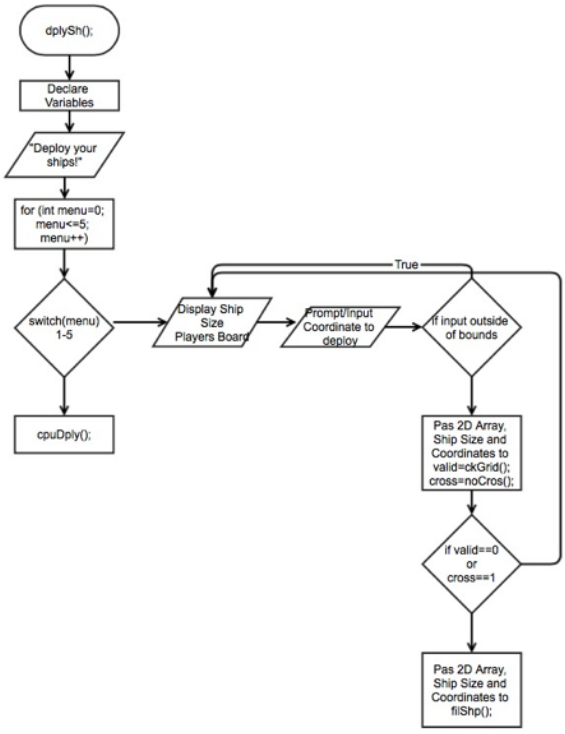


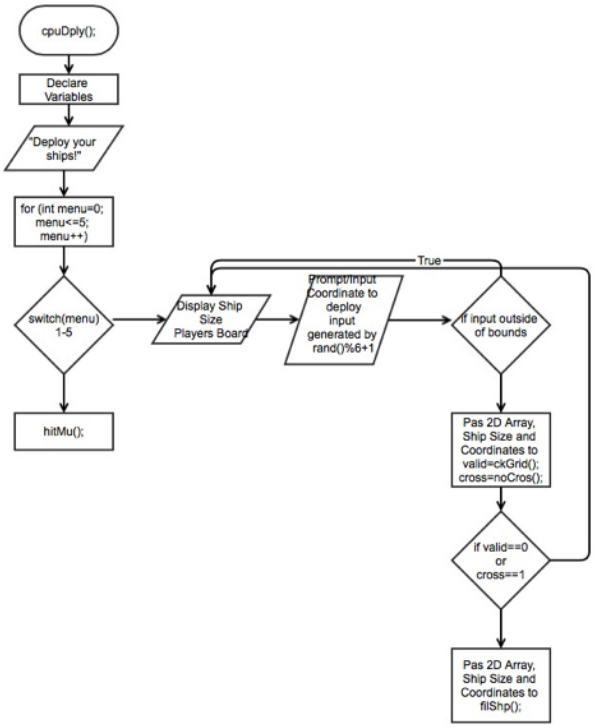
David Haro

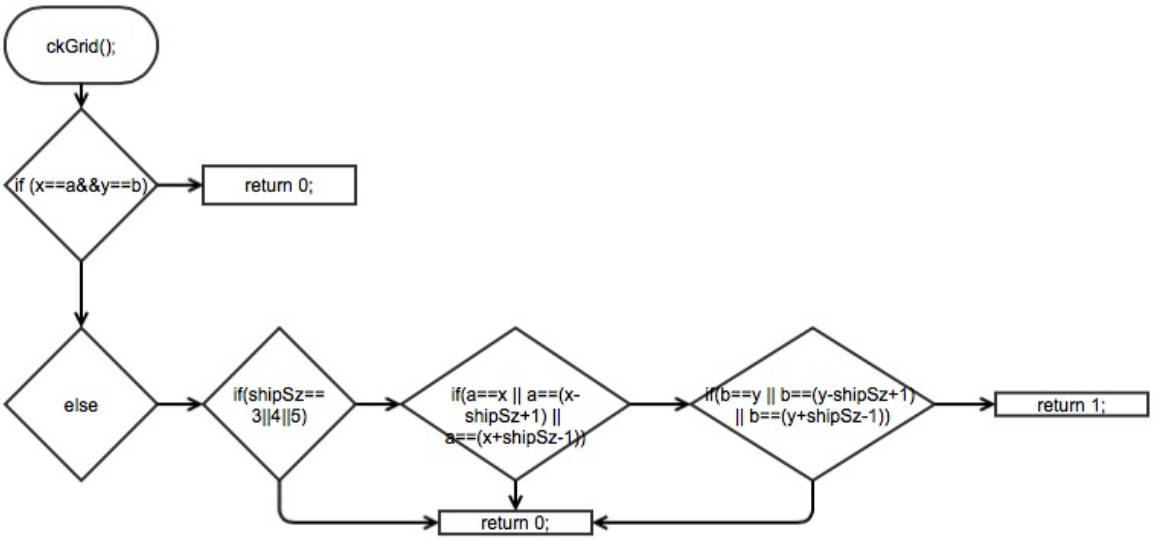
FlowCharts

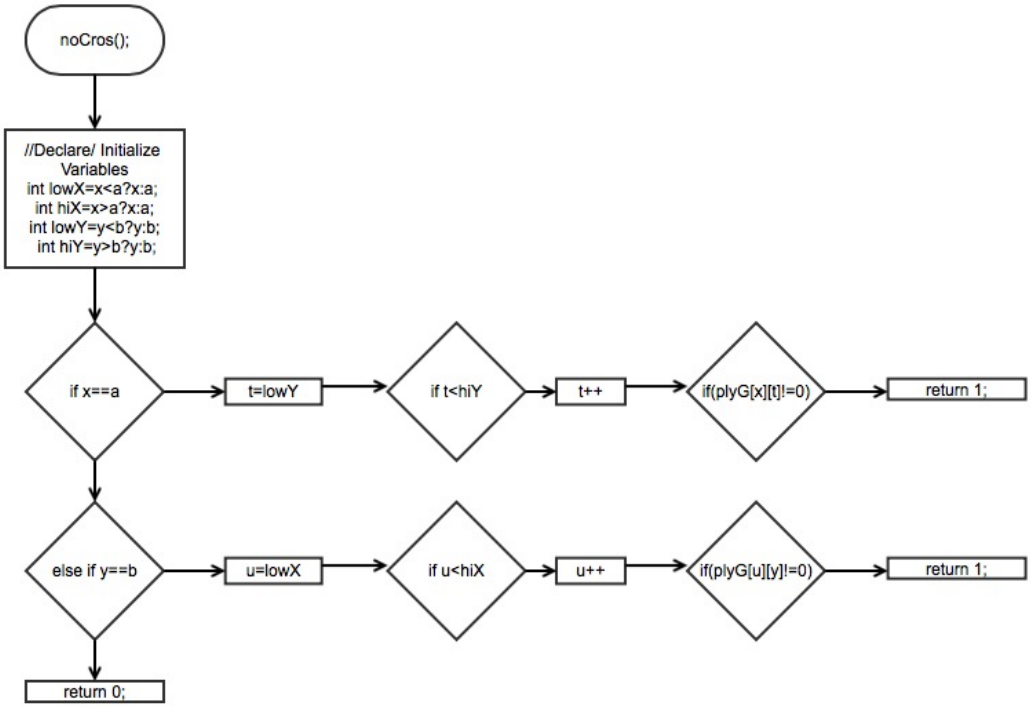


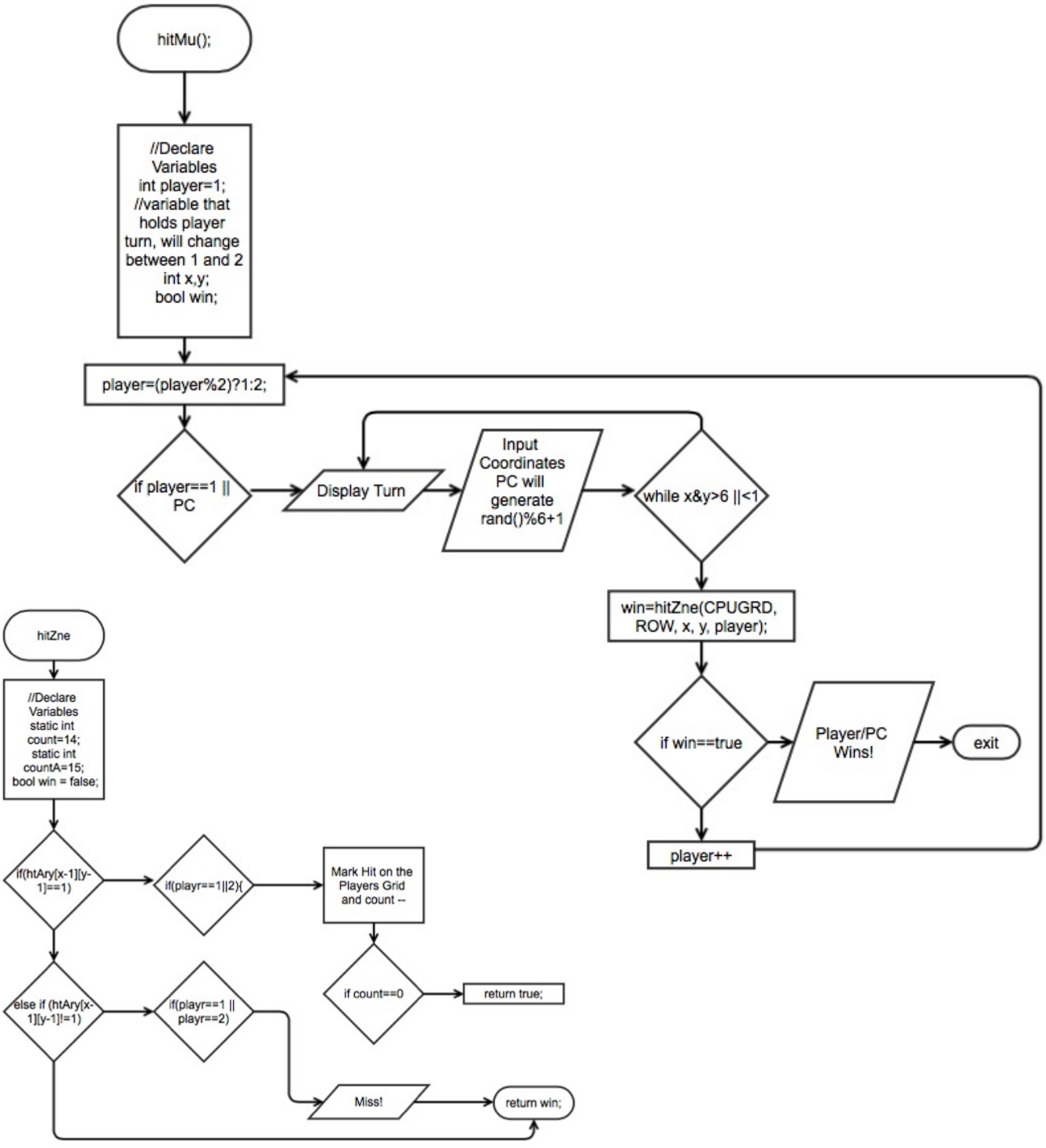


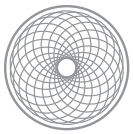





ckGrid();
if (x==a&&y==b)
return 0;
else
if(shipSz==3||4||5)
if(a==x || a==(x-shipSz+1) || a==(x+shipSz-1))
if(b==y || b==(y-shipSz+1) || b==(y+shipSz-1))
return 1;
return 0;


noCros();
//Declare/ Initialize Variables
int lowX=x<a?x:a;
int hiX=x>a?x:a;
int lowY=y<b?y:b;
int hiY=y>b?y:b;
if x==a
t=lowY
if t<hiY
t++
if(plyG[x][t]!=0)
return 1;
else if y==b
u=lowX
if u<hiX
u++
if(plyG[u][y]!=0)
return 1;
return 0;

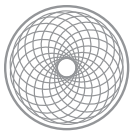

hitMu();
//Declare Variables
int player=1;
//variable that holds player turn, will change between 1 and 2
int x,y;
bool win;
player=(player%2)?1:2;
if player==1 || PC
Display Turn
Input Coordinates PC will generate rand()%6+1
while x&y>6 ||<1
win=hitZne(CPUGRD, ROW, x, y, player);
if win==true
Player/PC Wins!
exit
player++
hitZne
//Declare Variables
static int count=14;
static int countA=15;
bool win = false;
if(htAry[x-1][y-1]==1)
if(playr==1||2){
Mark Hit on the Players Grid and count --
if count==0
return true;
else if (htAry[x-1][y-1]!=1)
if(playr==1 || playr==2)
Miss!
return win;



David Haro

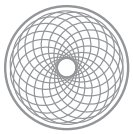
The Game: Broken Down

Type	Name	Description
Library	iostream	allows to input and output data
	cstdlib	c standard library: allows to shorten code and references C language for code.
	iomanip	allows to format output
	ctime	needed for random number generator
constant int	ROW	needed to initialize number of ROWS in 2D array
	COL	needed to initialize number of COLS in 2D array
int	HITGRID[ROW][COL]	2D Array for Viewing Players Hit Markers
Functions	int main();	Initial function. Purpose: Call strtMu();
	void strtMu()	Displays Name of Game, provides menu for play, rules and quit. Allocates and initializes (2) 2D arrays and passes to function calls.
	void bsBoard(int **);	Receives dynamically allocated 2D array then displays Player's Grid.
	void cpuGrid(int **);	Receives dynamically allocated 2D array then displays CPU's Grid if player hits a ship part.
	void dploySh(int **, int **)	Function that Deploys ships



David Haro

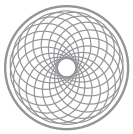
Type	Name	Description
	`int rules()`	Function for Game Rules
	`bool ckGrid(int **, int, int, int, int, int)`	Checks to make sure coordinates are in range, returns a boolean value.
	`void filShp(iint **, int, int, int, int, int)`	Fills a series of elements for respective ships.
Functions (cont.)	`bool noCros(int **, int, int, int, int)`	Check criss crossing ships. Returns a boolean value.
	`void cpuDply(int **, int **)`	Generate CPU ship grid with 2D dyna-mem array
	`int hitMu(int **, int **)`	Toggle player turn/Hit Menu
	`bool hitZne(int **, int, int, int, int)`	Checks if coordinate is a valid strike zone. Returns a boolean value
	`void hitGrd()`	Blank Grid to mark Hit Spots for Player to view
	`void destroy(int **,int)`	Receives both dynamic 2D arrays and destroys them prior to ending game.
Structure	`struct Legend{ string lgnd; int hitMrk; int ship;};`	Created to hold the legend in the display of any grids containing player/CPU hits to the left go the board.
Classes	`class Hit { private: string err; public: Hit(); virtual ~Hit(){} void setErr(); string getErr(); virtual void getHit(); }`	Class is created to display a graphic when player guesses correctly when asked to select a zone to hit on the opponent's player board. This, in the simplest terms outputs an ASCII graphic displaying HIT



David Haro

Type	Name	Description

Classes (cont)	class WBHit : public Hit{
	private:
	int err;
	
	public:
	WBHit();
	bool isRead();
	void getHit();
	};
		Similar to Hit class, this class inherits Hit and overrides the getHit function. Rather than just outputting the ASCII graphic via a cout, it reads the graphic from a binary files and outputs the contents.



David Haro

Improvements since previous version (BattleShip_v2)

* File: main.cpp

* Author: David Haro

* Battle Ship / Salvo Game

* Last Update for BattleShip_V2_5: Oct 26th, 2014 9:05pm

* Current Version: BattleShip V2_5_1: December 8th, 2014

*

* Updates Since version (BattleShip_V2):

* - Added Comments to function declarations

* - Expanded battlefield board from 6x6 to 10x10 2D array

* - Added player 2D Grids to be carried throughout the program

* - Fixed issue where last row will error when checking for cross

* - Added Legend to be displayed with player grid

* - Structure added to hold Legend diagram

* - Delete allocated memory before ending game

* - Fixed issue where CPU would only deploy to a 6x6 playing field

* - Squished other bugs!

*

* Updates for version BattleShip_V2_5_1:

*

* ->fstream read binary files.....DONE-----> rules read from binary file

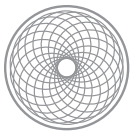
* ->classes.....DONE-----> class created for graphics (reads from binary file)

* ->inheritance.....DONE-----> WBHit class inherits Hit class

* ->dynamic arrays.....DONE-----> 2D dynamic arrays used for holding cpu and player grids

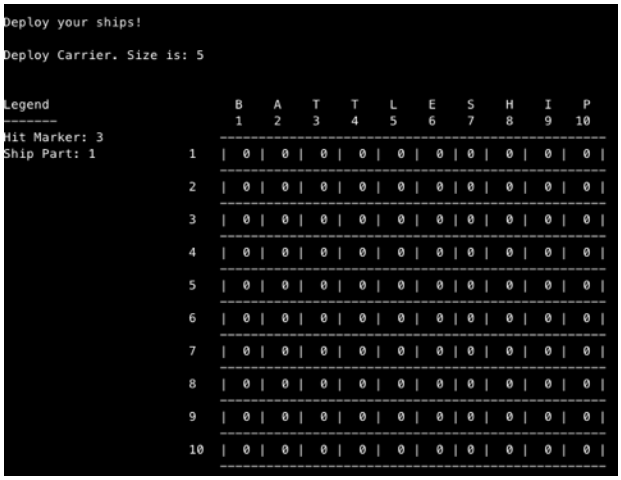
* ->polymorphic behavior.....DONE-----> overridden getHit method in WBHit class

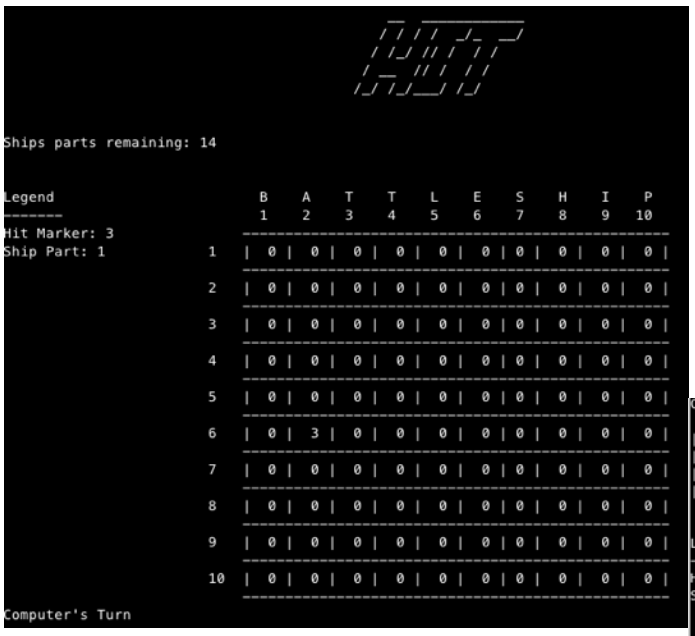
* ->templates.....Not Found-------> Trouble with setting up template class resulted in failed builds



David Haro

Screen Shots





<—-Player Screen when hit CPU ship

Graphic displayed when CPU hits player's ship location———->