**Design Document**

1. Title  
   A short program **title**, the **author's name**, **author's ID**, and **date**.

Battleship

Jake Billings

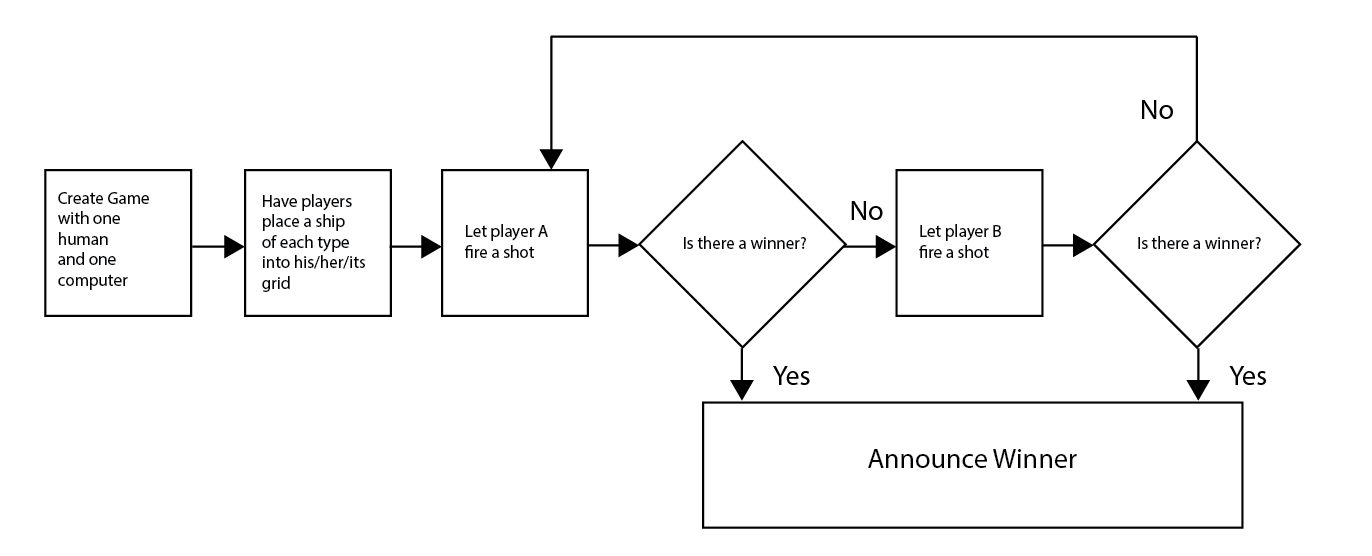
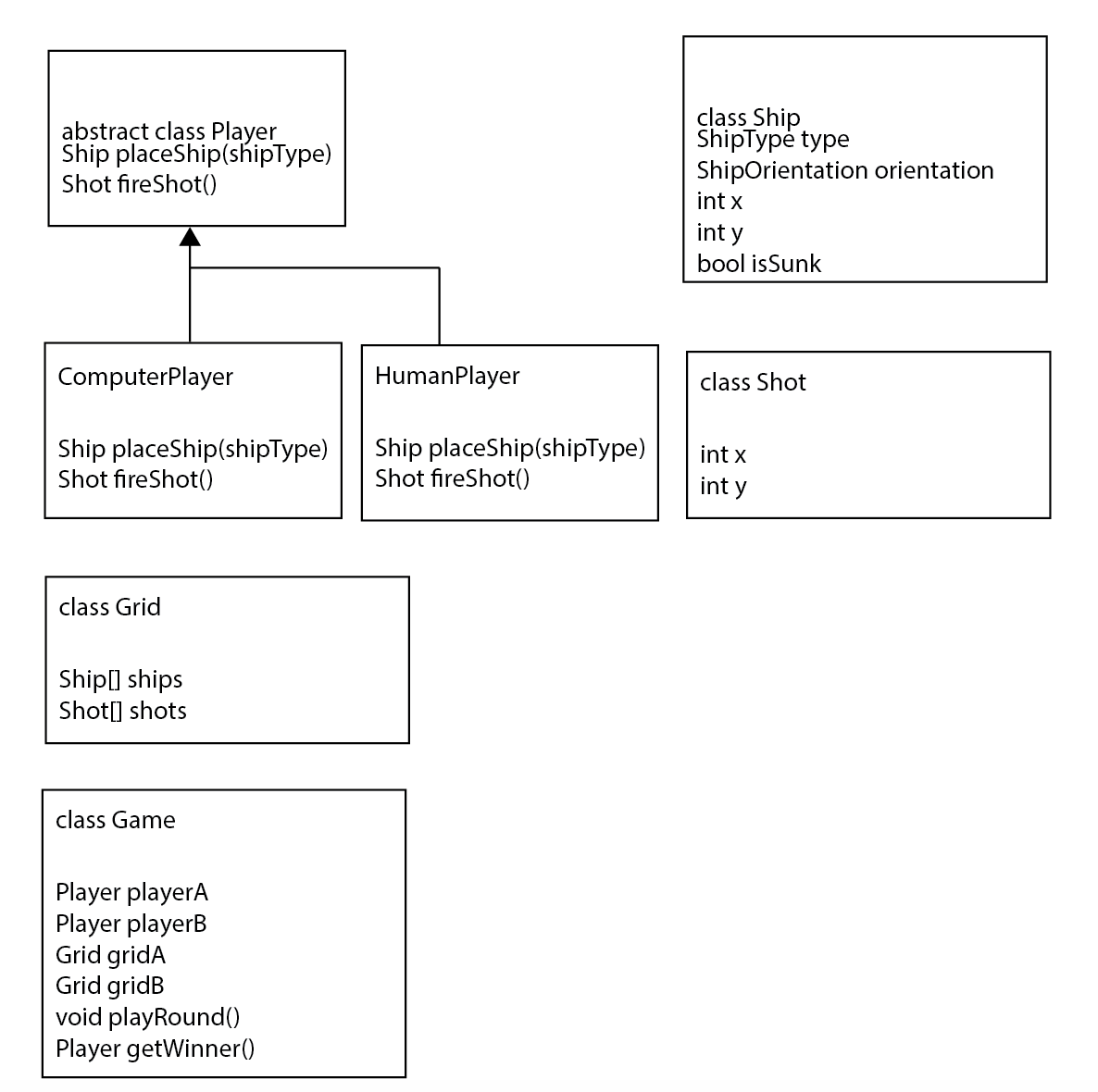
105955110

Feb 28th, 2018

1. Problem DescriptionA brief description of what this program does

Create a command-line-based game of battleship. Battleship is a game in which players secretly place ships on a grid then take turns guessing grid locations or “firing shots” until one player has all of his/her ship locations guessed (or sunk). The grid should be 10 x 10. A human player should face a computer opponent.

1. Overall Software Architecture

             
****

I plan to implement the software as a series of abstract data types (ADTs). Effort is made to minimize memory usage by storying ships in a “grid” as a sparse array structure of coordinates. The root object of a round of battleship will be the Game object. Game will store four sub-objects. Player A, Player B, Grid A, and Grid B. Game will contain logic for taking turns, checking winner, and announcing winner if there is one. “Player” is an interface for accepting input into the game. Players are able to place ships and fire shots. Computer and human players will exist. It is theoretically possible to have the computer fight itself, have two humans fight each other, or have a human fight a computer. The game stores the grid objects because player objects are intended to manage only UI/AI interactions. Grids contain ship location and logic to determine if all/some ships have been sunk. This will determine if a player has one the game. Ships and Shots will be classes containing relevant data.

1. Input RequirementsA detailed **list of all external inputs** (from files or keyboard) including a description of the **data type** and **range of valid values** for each input. For input file format and interactive user input, you need to write what data type is used for every field and valid value and length.

* FOR EACH SHIP A PLAYER PLACES:
  + SHIP ROW: unsinged int, less than grid height
  + SHIP COL: unsinged int, less than grid width
  + SHIP ORIENTATION: bool (should ship be horizontal)
  + COMBINATION CANNOT RESULT IN OVERLAP WITH EXISTING SHIP
* FOR EACH SHOT
  + SHOT ROW: unsinged int, less than grid height
  + SHOT COL: unsinged int, less than grid width
  + CANNOT RESULT IN OVERLAP WITH EXISTING SHOT
* FOR EACH GAME
  + Would you like to play again: bool

1. Output RequirementsA detailed **list or description of all outputs (**to files or to the screen).

Output: Visible Game Board State

* Player’s board
  + Where ships are placed
* Computer’s board
  + Where player has fired shots;
    - Shots are labeled as hit or miss

1. Problem Solution Discussion  
   A summary description of the solution steps with algorithms analysis (1 paragraph, approximately 100 words). If any unusual techniques or algorithms are used that need further explanation, and additional paragraph may be used.

Game play logic is fairly straight forward. Players take turns firing shots into each other’s grids until one player’s ships have all been sunk. The other player then wins. However, since I store ships in a memory-compact way, collision checking is computation-intensive. Ship inputs are validated for ship-overlap be comparing each ship to every other ship and checking if the rectangles overlap. Shot hit/miss is checked by comparing a shot to every ship and every location on the ship. Ship sink checking is performed by comparing every spot on every ship to every shot fired into the grid.

1. Classes, Inheritance, and Data Structures A description of choice of classes, inheritance, and data structures including your justification for each. Include a brief explanation for your choice. For example, "I have considered DS1, DS2, and DS3. Their pros and cons are summarized as follow... I choose DS1 over DS2 and DS3 because ...."

Some ship data is stored in enums. Ship type, which encodes ship length, is stored as an enum. This is an efficient way to store the type of a ship and imply its properties without the overhead of extending the ship class. A single int-like value in memory implies a name, abbreviation, and length for the ship. Orientation is also stored as an enum. It is stored as enum to be more explicit than a Boolean, which also would have worked. Ships are stored as a combination of X, Y, type, and orientation. This is the most memory-effient way I could determine. Other implementations such as chars on a grid may have made collision checking more efficient; however, data manipulations would have been harder and less memory efficient. “Grids” store vectors of ships and represent a player’s grid. “Games” store two grids and two players. They contain logic for taking turns. This is a “nice” way to encapsulate the turn logic. A flat function would work, but it would be less clean. A standard input interface for player actions such as firing shots and placing ships will be used. This will allow for abstract handling of turn logic while allowing for any combination of human and computer players to start a game.