Objectives
Create a Turn-based Strategy game, based on the game battleships. should:
- Display a StartMenu, LoadMenu, Instructions and quitting.
- In game, it should prompt for a coordinate, try attack that coordinate and return the result of the attack
- Allow users to input 5 unique locations
- Render the Grid with boats
- Displays a "Target Tracker"
- Create a "Computer" that can play against Humans
- Plays again when game end

Challenge:
- Save Game Each Turn(serialize/deserialize JSON)
- More Boats

Documented Design
Structs & Enums
the Player (Human or Robot) Stores a Player grid (pGrid) and an Enemy grid (eGrid), which are a 8x8 2D array of Tiles. Players also store their "name" which is input by the user if the Player class is inherited by a Human, or "robot" if it is a robot.
The Tile enum contains:
- `Hit`: which represents when a boat has been hit
- `Missed`: which represents an empty tile which has been shot
- `Empty`: represents an empty sea tile
- `v_S,h_S` : represent the vertical and horizontal first character of the ship
- `v_E,h_E`: represents the vertical and horizontal end character of the ship
- `v_N,h_N`: represents the vertical and horizontal middle part of the ship

The Ship struct contains:
- String `name`, the name of the ship
- int `amount`, how many of this ship exist
- int `length`, how long the ship is

The SaveFile struct contains:
- Player1′s player and enemy grid and name
- Player2′s player and enemy grid and name
- the "type" of game, PvP/PvE

The FgColours and BgColours are enums which contain ansi colour codes.

Coordinates are interchangably either a tuple or two separate variables depending on function use.

Functions

Main
Inputs: `args`
Returns: `null`
the entry function
Game.init
Inputs: `null`
Returns: `null`
renders all the menus, and allows for moving around
Game.init_PvP

Inputs: `null`
Returns: `null`
generates all the data for a PvP game

Game.init_PvE
Inputs: `null`
Returns: `null`
generates all the data for a PvE game

Game.turn
Inputs: `Player, Player`
Returns: `boolean`
runs a player "turn" and checks for losing condition

Game.PvP
Inputs: `null`
Returns: `null`
plays the game player vs player until a player wins

Game.PvE
Inputs: `null`
Returns: `null`
plays the game player vs entity until a player wins

Game.Serialize
Inputs: `string`
Returns: `boolean`
serializes the gamestate to a json specified in path

Game.Deserialize
Inputs: `string`
Returns: `string`
deserializes the game from the path, and returns the type of game

Player.init_grid
Inputs: `null`
Returns: `null`
generates an empty grid

Player.Attack
Inputs: `int,int`
Returns: `bool`
checks players grid and returns if the tile is a ship

Player.result
Inputs: `bool,int,int`
Returns: `null`
registers the result to the player

Player.hasLost
Inputs: `null`
Returns: `bool`
returns if the player has lost

Player.format
Inputs: `string`
Returns: `(int,int,bool)`
formats a players coordinate input, e.g. "a4,4a" into a coordinate axis, and responds on if it was a valid coordinate

Player.render_grids
Inputs: `null`
Returns: `null`
renders the player and enemy grid
Player.verify_placement
Inputs: `int,int,Ship,bool`
Returns: `bool`
checks if you can place the ship in the orientation and position, if possible places, else returns false

Component.w
Inputs: `string,FgColour,BgColour`
Returns: `null`
“writes” to the current component with Colours
Component.wl
Inputs: `string,FgColour,BgColour`
Returns: `null`
“writes” a line to the current component with Colours
Component.draw
Inputs: `int,int`
Returns: `void`
renders the component at the position

Human.Turn
Inputs: `null`
Returns: `(int,int)`
performs a humans attack
Human.place_ships
Inputs: `Ship[]`
Returns: `null`
places all the ships with player input, error handling

Robot.Turn
Inputs: `null`
Returns: `(int,int)`
Performs Robots random attack selection
place_ships
Inputs: `Ship[]`
Returns: `null`
places all the ships randomly, error handling

Classes
- Player: abstract class which contain shared code between Human and Robot classes
- Human: inherits Player class, allows for user input of ship placement and attacking
- Robot: inherits Player class, acts like a robor
- Game: plays the game for you, also acts like a menu handle
- Component: General component rendering/creation
- Components: Pre written components used for UI

Design Decisions of Interest

Player.hasLost()

```
!pGrid.Cast<Tile>().Any(tile => (tile != Tile.Hit && tile != Tile.Missed && tile !
= Tile.Empty));
```

As I had multiple characters for Ships, i checked if any of the tiles were Not not a ship, cleverly using the cast method to flatten the 2D array, and .Any() is at worst case O(n) but does break once a “true” condition is found

File Serialization/Deserialization

Done through Newtonsoft.Json as it allows for simple and clean typecasting without weird hurdles.

Rendering

as the rendering of the grid had spacing in between to create a more equal looking board, my rendering function contained code which replaced spaces with the correct character to make boats seem more full

Ship Rendering

my ships have separate characters for start, ends and centers, depending on orientation and length of ship

the entirety of the Interactions between players

Due to lack of foreshadowing, a seemingly good idea turned into a caffeine fueled little problem which was then fixed by more caffeine fueled little problems, it works but it is not what was intended

Evaluation

Overall, the project achieved the outlined outlines whithin a reasonable accuracy, including the further challenges, if the project would be undertaken again. i would

- rewrite it in rust
- make the overall UI nicer to use
- plan the entire project structure beforehand
- use more descriptive variable names
- make the AI more