**COMP813 Artificial Intelligence, Semester 2, 2023**

**AI Project Option A: Battleship Game**

*Finn Boyle, 18034590*

*Introduction:*

For this project, I have created a game of Battleship that is playable against an Artificial Intelligence (AI) opponent. This project uses supervised learning, heuristic searching with elements of knowledge representation and was built nearly entirely by me alone, with some assistance in creating the AI from Coding Cassowary’s Battleship game tutorial series on YouTube **[1]**. I chose to build the project in Python, the game is text based and is played entirely using keyboard inputs.

*Problem Definition:*

To quickly define Battleship, it is a strategy game played without perfect knowledge of the game state. Players have two boards each representing friendly and enemy seas, hidden from their opponent, the friendly board is for placing your own ships, and the enemy board is for recording shots taken against the opponent. Players place their five ships of lengths five, four, three, three, and two on their placement grid. The aim of the game is to take turns “shooting” at co-ordinates on the opponent’s board. The opposing player confirms if the shot hit or missed a ship. If the shot hit, the shooter will mark their enemy board in the hit location with a red pin, and the opponent marks their friendly board with a red pin. A white pin is used the shot missed. This continues until one player has sunk every one of their enemy’s ships, and this player is the victor. This is a slightly modified version of the official Hasbro Battleship game rules **[2]**. Rule modifications made include players not needing to state hit/sunk ship types – forcing both player and AI to search the board more rigorously. Another rule modification is the need for two visual boards each. Instead of a ‘ship’ board and an enemy ‘sea’ board, the AI uses the player’s board to record its hits. As such, the player has their normal enemy board but shares their placement board with the AI. The AI cannot determine anything about the player’s placement board except for where hits have been recorded. I felt this would save resources, due to the AI not truly being a real player and does not need to store another board for its own enemy seas when it could have instead very limited visibility of the player placement board.

The main problem with teaching an AI to play Battleship is letting it learn optimal shot placements outside of firing next to other hits. The AI can search the board randomly, but this isn’t competitive. Firing randomly, the AI will usually require the maximum 100 shots to win, as evidenced in a study by DataGenetics (2011) **[3]**. To improve, the AI can fire in a checkerboard pattern, searching every other tile, as the smallest ship is of length two. DataGenetics **[3]** found that this method nearly halved the number of shots needed to 60.

Nearby hit searching, random and checkerboard targeting were readily implementable. So I expanded on these methods with an AI that would be able to determine shot placements from data collected in previous games – i.e., the AI would be able to predict human thought, learning where humans prefer to place their ships in Battleship to increase chances of winning. I needed a way to let the AI learn where humans would place ships, such that it could be a challenge to the human player.

*Motivation:*

I chose Battleship for this project because it was something new and challenging for me. Also, as I began researching the project, it seemed that no other Battleship AI was implemented in the way I propose. Instead, many projects focused on single game learning, where the AI would make the most optimal shot based on the current game state instead of historical game states. Therefore, my final motivation was to create an AI that could improve after every individual game.

*Methodology:*

To build my Battleship game, I used the following structure of discovering requirements, developing, and testing the game and its corresponding logic, developing and testing a basic AI, and finally implementing the full intelligent AI.

1. *Requirements discovery.*

Before I started on anything, I tried to get an initial idea of what I would need. First, I needed a game for the AI to be placed into. I started brainstorming required Python functions. These initially included two boards per player (one to show and one for processing), and functions for printing the public versions, as well as ship dictionaries for each player including name, symbol, length, hits needed to sink, and a sink check. Other functions included ship placing, shooting, and hit, sink and win detection. Sanitisation of player inputs was also required.

1. *Game board and logic development.*

Each mentioned previously mentioned feature is in game. I considered a third set of boards per player for processing shots to opponent boards, but these were scrapped.

The game began by allowing the player to place their ships in any order and location either horizontal or vertical on the board. Next, the AI ships would be placed randomly. After this, participants would alternate shots at each other, until a win was recorded, where the code would finish.

1. *Testing the game board/logic.*

Of course, the game building was not without issue, much testing of the game logic was needed before any AI could be implemented.

The first challenge was to allow any ships at all to be placed. Bugs included the ability to lay ships on top of one another, placing ships out of bounds, and illegal player input. Each of these problems was solved before moving on. At this point, AI placement was not implemented.

Next, testing the shooting function. For this, I aimed the shot function at the player board and fired upon my own ships, allowing me to focus on the shooting without AI interference. Shots needed to change empty sea co-ordinates from “~” to “M” to indicate a miss and needed to change any occupied locations from ship symbols to “H”. Repeated location shots were removed by disallowing shots containing an “M” or “H” symbol, forcing a retarget. Board updating was successfully implemented. After satisfactory testing, targets were reset to player-to-AI and AI-to-player.

1. *Basic AI development.*

Thanks to YouTube’s Coding Cassowary **[1]**, I implemented a basic AI into the game. This AI shoots across the board in a checkerboard pattern, checking every other co-ordinate for ships (as the destroyer is the smallest ship, of length two) otherwise it would shoot randomly after exhausting the checkerboard. If the AI registers a hit, it begins searching neighbouring locations for more of the ship. If another hit registers in a neighbouring location of the original hit, the AI then begins searching for further hits in a line extending out from the first two hits until no further hits are detected.

I extended this AI with checkerboard upgrades. If the aircraft carrier (length five) is not sunk, the AI now searches every fifth co-ordinate, as this ship length is still discoverable in this pattern size. If the carrier is sunk, the battleship – length four – is selected and the AI checks every fourth location. The checkerboard is reduced as larger ships are sunk, until only the destroyer is left. Both the submarine and cruiser need to be sunk before reducing the checkerboard pattern, as each is length three. I retained the random shot function if no checkerboard spots remain, although it would be seldom used. Additionally, Coding Cassowary’s AI was built for a program that displayed the game in a GUI (Graphical User Interface), which I adapted to instead work in my text-based format.

In this section, I also implemented a method for the AI to place its ships randomly across its board. This method was nearly satisfactory from first testing, as it is forced follow to the same placement rules as the player, it could only place ships in unique locations without overlap, and within the game bounds. The only issue was that I needed to make the function print placement warnings to the player only, as the AI would repeatedly place ships until a legal configuration was discovered.

1. *Testing the basic AI against players.*

I tested this AI by simply playing against it, tweaking values as needed. The game was playable, if simple. The AI shot at the player board with some strategy, an improvement on random shooting, but couldn’t learn from the played games. It still could not guess player placements.

1. *Improving the AI with supervised learning methods.*

With a playable game and backup AI, I could make it learn. To do this, I created functionality for reading from and writing to an external file that would contain accumulating previous game data. After placing their ships, player ship data is saved to a Comma-Separated Values (.csv) file. Placement data is saved as a 10x10 grid, ones representing a ship presence, and a zero for a lack of a ship, this information is then added to any information saved in the file.

With saveable data, I could begin improvements on the AI targeting. Before either player shoots, save data is read into a hidden board from the training dataset and translated. Each value in the new board is replaced with a weighted value – the co-ordinate’s integer value divided by the maximum integer value in the training data. This weighted grid allowed the AI shot function to prioritise shooting at common player placement locations – with an 85% certainty value to allow for outlying placements. The AI will add any locations with a high enough certainty to its potential shot list and make shots onto the board from that list – after which it removes that location from the list of valid shots. Should there be no locations above the certainty mark, the AI may lower the required certainty value.

If this function fails, the AI reverts to the basic AI, which will in turn revert to random shots if needed.

1. *Training and testing the improved AI.*

Finally, testing of the improved AI, and accumulation of training data. Since I save ship placement data from the current game before any shots, the first test game meant the AI was 100% guaranteed to hit on its first shot. However, this advantage no longer exists, as the training data has built up enough to make this advantage negligible. To increase the rate at which the AI learned, I temporarily stripped functionality from the game and repeatedly input ship locations, where the game would terminate and allow me to repeat the process, saving those new locations once again. Each time, I placed ships in locations I thought would be tactically secure, to increase my chances of winning.

After increasing the AI’s knowledge dataset, I returned the game to full functionality, and played normal games against it to gauge its intelligence. The AI will technically never stop learning, but its probability database will plateau with enough played games – the value of adding one to cells where a ship is placed will have continually decreasing effects on weighting as game numbers go up.

*Evaluation:*

After creating and testing the Battleship game and AI decision making process, my AI can be classified as something that learns over time and is trainable – it uses a form of supervised learning. The player can play against the AI – the player is able to choose locations to fire at, and the AI is able to determine locations to return fire based on accumulated game data. Furthermore, as the player and AI fire at each other, they can sink each other’s ships, the first one to sink all the other’s ships wins. The game is functional, with a computer acting as the player’s opponent. This computer can learn from each played game how a human player thinks.

My Battleship is entirely text based, therefore with limited visual options, but I believe it is enjoyable. The AI is a good opponent, although it is susceptible to bias – especially when trained to play against one player specifically – when new players try the game. The AI does have some quirks, it will search every co-ordinate neighbour of a hit, hoping another ship as placed side-by-side with the original. I chose to leave this in, however, as I was tricked by the AI’s ship placements in some of my test games, I had thought one ship sunk, when it was really two ships placed directly next to each other.

In terms of the type of AI, my Battleship game incorporates supervised learning, heuristic searching, and some knowledge representation with the board of likely ship placements.

Supervised learning, as defined by IBM **[4]**, “is defined by its use of labelled datasets to train algorithms that classify data or predict outcomes accurately”, which my algorithm makes use of. Every cell in the weighted grid is a likelihood of the corresponding game grid location containing a part of a ship. The training data file is a collection of real game ship placement datapoints and is used to make decisions on where to place the next shot. The algorithm also incorporates heuristic searching to help it determine where to place any given shot. Heuristic searching being defined by Randall (2020) as “searching the solution space while assessing where in the space the solution is most likely to be and focusing the search on that area” **[5]**.

With the weighted probability board, my algorithm uses a simplified form of knowledge representation – defined as storing data in a database and learning from it over time such that it may become more intelligent over time **[6]**. The weighted board is a historical, numerical record of all previously played games. The AI is able to use this record to store new information and draw on old information to make decisions on currently occurring games.

*Discussion – Limitations:*

This project was certainly not without limitations, major ones being my order of feature implementation and difficulty in finding premade datasets of common Battleship placements.

If I were to do this project again, I would change the order of implementation of features. My problem is that I created the game logic before creating the AI opponent. Rather than building the game around the AI, I was building the AI around the game. This limited what kind of AI I could implement into the game. Initially I had planned for the AI to use Monte Carlo Tree searching and reinforcement learning, but my game functions were not prepared to handle this type of searching and no functions were equipped to give rewards/punishments to the AI for good or bad actions. As such, heuristic searching, and supervised learning were the next best options that would fit into my game logic without requiring entire redesigns of the system.

Secondly, the datasets I was looking for do not appear to exist, at least not in the form I need. For my system to work, I need a grid of most common human ship placements that maximise – in the human’s perspective – the chance of winning. In the end, I could not find any dataset such as this, and I could not resort to automatic generation of the dataset as players needed to manually put in ship placements so I could build up a dataset of the human experience in placing ships. Instead, I trained the AI manually, having the game played many times – full games and altered games where only the ships could be placed – to build the dataset on my own.

*Discussion – Future Work(s):*

If further work were to be done on this project, my primary suggestions/ideas would be to implement a system for the AI to learn where it could best place its own ships, a method to improve the AI such that it ‘gives up’ on an area of the board after finding no extra hits near a previous recorded hit – as mentioned earlier in this report, and finally to allow the AI to switch strategy mid-game.

Currently, my AI has its ships placed randomly. This project could be improved by making the game more difficult for the player, as the AI could learn where a player is most likely to take shots against it and avoid placing ships in areas players commonly shoot at.

Furthermore, the current iteration of my AI will surround a ship with misses after getting all hits on it. For an example, if the aircraft carrier is hit, the AI will hit the first part, then work its way through the rest of the ship, but once the entirety of the carrier is hit, the AI will continue in that area. It will continue to take shots directly on either side of the ship, and at its tips, chancing that the aircraft carrier is really the battleship and one hit of another ship. I would improve upon this issue by allowing the AI to give up on an area for a time if there have been many unsuccessful attempts at this specific area already. If no further successful hits occur elsewhere for several shots, the AI may return to this area.

Finally, I would allow the AI to alter its strategy mid-game. Currently, if the player knows or has access to the information in the dataset, the player would be able to outsmart the AI, placing ships in areas statistically least likely to contain a ship. I could allow the AI to invert the weighted board for a small number of shots if all previous shots result in no hit for the AI, letting it assume that the player might have access to its knowledge base.

*References:*

1. Coding Cassowary. (2022, April). *Battleship AI*. YouTube. <https://www.youtube.com/playlist?list=PLBLV84VG7Md8SgHlXuQXPMJLDvCaWVVQv>
2. Hasbro. (1990). *Battleship*. <https://www.hasbro.com/common/instruct/battleship.pdf>
3. DataGenetics. (2011, December 3). *Battleship*. Retrieved October 6, 2023, from <http://www.datagenetics.com/blog/december32011/>
4. *What is Supervised Learning?  | IBM*. (n.d.). <https://www.ibm.com/topics/supervised-learning>
5. Randall, M. (2020, April 1). *Heuristic Search Part 1: Introduction and Basic search Methods – Matt Randall*. <https://www.lancaster.ac.uk/stor-i-student-sites/matthew-randall/2020/04/01/heuristic-search-part-1-introduction-and-basic-search-methods/>
6. *Knowledge Representation in Artificial Intelligence - JavatPoint*. (n.d.). www.javatpoint.com. <https://www.javatpoint.com/knowledge-representation-in-ai>