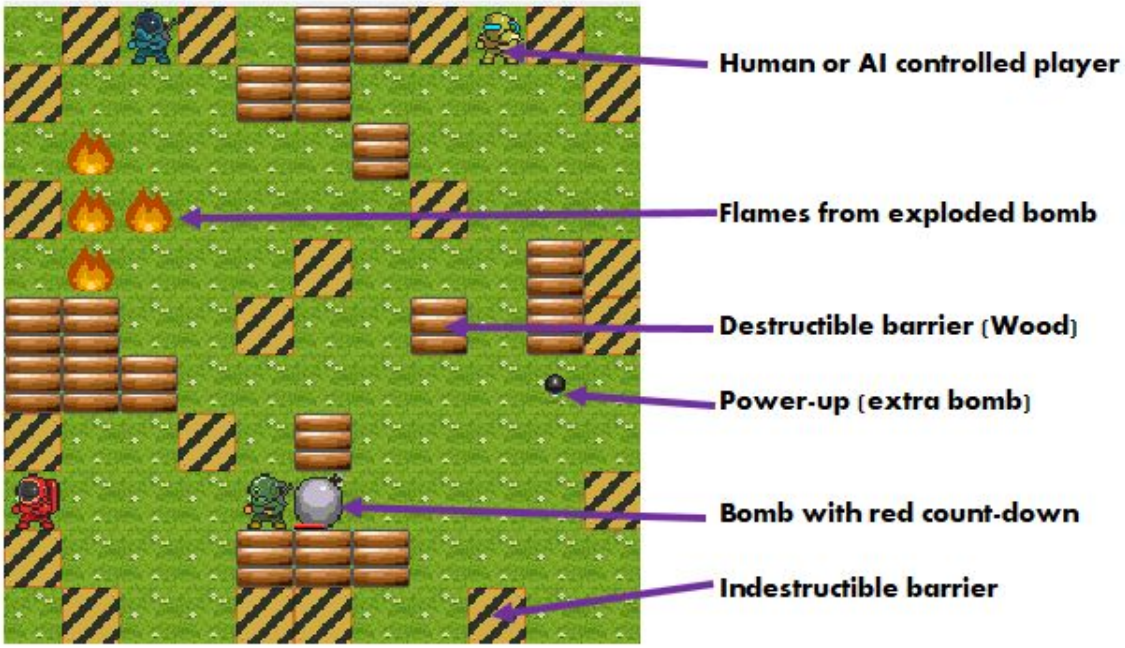
Concept and Introduction

The base Pommerman game involves four players moving around a map grid in real-time, planting bombs that explode after a certain number of game-ticks, and trying to be the last one standing. Power-ups can be picked up for additional bombs, to allow bombs to be kicked across the map, and to increase the blast radius of your bombs. (See diagram below.)
The concept for this project is to augment the game with an element of negotiation between the players, and use this to influence the AI behaviour of agents in interesting ways. The stakeholder need is player entertainment, and the key technology is heuristic functions in RHEA/MCTS with configurable utility weightings.



Pommerman is a real-time game played with the cursor keys (to move), and the space-bar (to drop a bomb). While there is a Pommerman-variant that allows AI agents to send messages to each other every turn, this would not work with human players unless we migrated to a turn-based set-up. We felt that Pommerman was not strategically rich enough for this and our initial design constraints were:

1. Keep the real-time nature of play;
2. Keep the same cursor/space-bar keys for human control during negotiation to reduce cognitive switching costs;
3. Ensure human and AI players have the same constraints and negotiation options.

An initial brainstorming session proposed the following structure:

- At 3-4 points in the game, we pause the main game-play for a number of ticks, and have a two-phase negotiation process.
    - In the first phase each player can send a number of proposals to other players.

    - In the second phase players can accept/reject the proposals received.
- Five types of proposal were agreed on:
    - ***Alliance***. This is a simple promise not to deliberately kill the other.
    - ***Stay Apart.*** The two players agree not to move closer than X squares to each other.
    - ***Share Vision.*** The two players agree to share perceptual fields; each can see what the other can see.
    - ***No Bomb Placing.*** The two players agree not to place a bomb if the other is within X squares.
    - ***No Bomb Kicking.*** The two players agree not to kick each other's bombs. (This was changed in implementation to not kick any bomb toward the other.)

The last four of these proposal types could be enforced in game mechanics, while the Alliance option would initially only be incorporated into AI agent's heuristic functions. This could be exploited by a human player, and a later iteration could address this. The remainder of this report document the development process and technical implementation of this concept,

Process

The process for making and improving the game started with the creation of an online ***miro*** whiteboard, where each element of the game could be displayed on its own sticky note, and proposed ideas could be attached for discussion at group meetings. The whiteboard helped prioritise ideas that were essential compared to those to return to if we had available towards the end of the project such as ideas for additional alliance types. The white board can be seen at:
https://miro.com/app/board/o9J_knYaSCE=/

The miro whiteboard helped to identify tasks that could be initially worked on independently and those which required communication between different parts of the game through the intended messaging system. This informed which tasks each member of the group took on, with each person recording their progress and outstanding things to complete on a ***Trello*** board which can be seen at:
https://trello.com/b/gFyMKmaC/pommertalk

These initial tasks that each group member took on were as follows:
James – Changes to core game logic. Heuristic changes for RHEA/MCTS agents.
Gideon – The Graphic User Interface (GUI).
Guilherme – Messaging communication. Implementation of Negotiation agents.

The group conducted a call on ***Discord*** every morning to discuss current progress, ask each other to test code changes and report what was required from each member to make sure each new addition to the game could be implemented successfully. Importing the original version of

Pommerman into the ***IntelliJ IDE*** allowed for good version control using ***git***. Each member of the group worked on their own branched version of the game through ***git***, regularly merging code that had been confirmed by the group to be working into a master branch. This helped avoid situations where code added to the master might inadvertently cause a game breaking bug and allowed for an iterative agile style development process to take place.

In addition to internal play testing throughout the game's development, it was also useful to get external opinions from other members of the IGGI group on the playtesting days. This helped identify an issue in compiling the game to an executable jar file, as well as the initial difficulties in playing the game for the first time as a human player. This realisation led to pausing the game whilst human players choose their alliances and letting them continue when ready, rather than being restricted to a pre-set countdown timer.

Technical Details

Code repository: https://github.com/hopshackle/Pommertalk

Core Engine changes

Working from an existing well developed code base restricted our architectural options given the project length and our starting unfamiliarity with the code. As far as possible we kept the negotiation implementation to new classes, and tried to minimise changes to the core game engine, and hence the risk of introducing new bugs to it.
The main core classes changed were:

- `Game.java`: The tick() method that advances the game state was amended to have a mini-Finite State Machine to shift between three phases, in order: NORMAL, NEGOTIATION_ONE, NEGOTIATION_TWO. (A PAUSED state was added after play-testing.) This maintains a link to a Negotiation class instance that handles the new functionality. Methods on this instance are called to handle all the negotiation phases.
- `ForwardModel.java`: The next() method that advances the details of the game state (either in planning simulation by a RHEA/MCTS agent, or in the actual game) was modified to call methods on the new Negotiation class to implement restrictions on actions due to player agreements (all except for the Alliance type).

A new negotiation package then contains the core code for the new functionality. `Negotiation.java` is called at each Negotiation phase, and invokes methods on each Negotiator in the game. `Negotiator.java` specifies a new interface with two methods to be implemented (`makeProposals` and `reviewProposals`). MCTS/RHEA agents were amended so that a Negotiator could be provided on instantiation, so that new Negotiation logic could be kept entirely separate from the existing code.
The final new core component is `MessageManager.java` in the new message package. This controls all communications between the GUI, the core engine and the individual AI agents.

GUI changes

The GUI required for the game had two main functions:

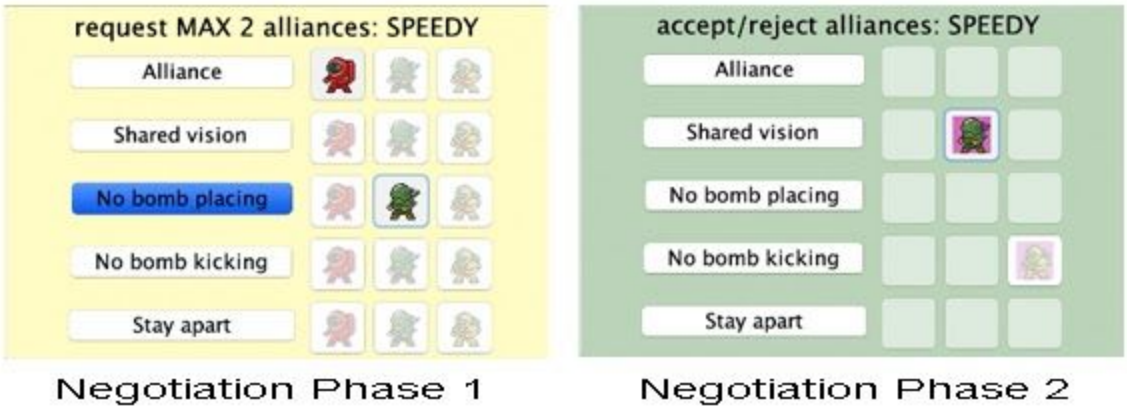
1. To allow a human player to request alliances with other AI agents, and subsequently to choose from the alliances received from those agents.
2. To allow a human observer to watch a game between four AI agents, seeing the different alliances formed at each stage between the different opponents.

Early discussions informed that the GUI should use the same keyboard interface when playing or as a human player or observing during the standard Pommerman game. This involved the ***arrow keys*** to move and the ***space bar*** to set a bomb. Additionally, it was decided to use the existing blank space underneath the main game screen to display the alliances currently in play and to make selections. This new display was created on a new Java panel, and through an array of Java toggle buttons as shown below that could be reused at each stage of the process.

Through highlighting the relevant buttons it is quick to select or identify if each of the alliance types exists or has been requested with each of the selected player's three opponents.. When selected both the rule and opponent icon are highlighted to display that this particular alliance is currently in play. Additionally, when watching a game between four AI agents, an observer can select any of the four agent characters, and the buttons will automatically update to show their personal set of alliances with each of their three respective opponents.

The information displayed to a human player or observer was iteratively changed over the course of the game development to make their available options more intuitive and remove the frustration from unfair deaths. This involved some additions to the basic Pommerman game such as displaying the time to the next wall collapse, since players on the edge of the screen when a collapse occurs would be killed without necessarily knowing a collapse was due. Each negotiation phase is clearly colour coded and human players also have an unlimited amount of time to make their alliance choices with the game paused, rather than being time limited like the AI agents. In a purely AI game, the alliances requested and chosen are displayed using the same two selection phases as a human player, making it clear what selections have been made by each agent.

In the first stage of the alliance negotiation a human player can make up to 2 alliance requests with the three agents on any of the 5 possible alliances. In the second stage they can only choose from the alliances they have received from their opponents. The keyboard listener events of the buttons change during each negotiation phase to provide speedy movement between the active buttons. Initially the available selections also flashed during the second negotiation phase, though this was removed when the game was paused since the game ticks no longer update, preventing its inclusion in its current form. This might be something to reinstate in a future version of the game.



Negotiation Phase 1 Negotiation Phase 2

The GUI directly connects with the message system during the negotiation phases to create an array of all the alliances formed between all players, incorporating the human player's choices if one is present. Keeping all player data in this way allows the GUI to be updated with any player's alliances after a single game tick, so that the human player is given the option to observe the other players' tactics after they have died.

Message System

The core of the messaging system is the Message Manager. This has the function of creating, storing, sorting and sending the messages, crucial to this addition to Pommerman. It was built with the idea that it can be converted into an online messaging system, which influenced the way messages are stored, as well its extra functionalities, like search and translation.

Recording Messages

Messages are made of 6 variables recorded in a HashMap. This is done to mimic the usual message format JSON. Four of the recorded variables are the ID (individual to each message), the agent which sent the message and its intended receiver, and finally the round in which the message is sent. The content of the message takes the other 2 variables: the proposal, an integer indicating which agreement is being discussed, and the response, which indicates if the sender intends to send a proposal, or respond yes/no about a proposal to the receiver.
As a note, the sender and receiver variables are always in the relation to the message. So if player 1 sends a proposal to player 2, player 1 is the sender and player 2 is the receiver. And when player 2 responds positively, player 2 is the sender and player 1 is the receiver in the response message.

Fetching Messages

It is possible to retrieve messages either by ID, used only for debugging purposes, or through searching for a combination of variables in messages. This allows us to retrieve messages for specific agents when searching for proposals, and match responses to find the agreements made during a round.

Translating

An important function of the Message Manager is translating information between the different parts of the PommerTalk game. As the GUI and the game negotiators require information in different formats. So, much like a messaging system, the manager works as a translator between the messages and the required format.

MCTS/RHEA Heuristics

By using different heuristic functions to evaluate the worth of a state reached at the end of a roll-out in either RHEA or MCTS we can obtain different behaviours using the same underlying algorithm. There were two goals here:

1. Incorporate the results of negotiated agreements into agent behaviour
2. Differentiate AI agents more generally in 'personality' in an easy to configure fashion.This was not an initial goal, but arose during the project.

The base Heuristic used from the original framework was `CustomHeuristic.java` which was hard-coded to like states with few living enemies, no wooden walls and with as many blast radius power-ups as possible. This causes agents to open up the map at the start of the game, and then focus on killing each other. For this project this was extended to include getting extra bomb power-ups and to care about one's allies. The relative weightings were pulled out of the class so that different agents could be easily configured to have different priorities. This can be seen as a form of 'Utility AI' in which we tweak the high-level preferences and let the underlying algorithm do the rest. The final vector to be set for each agent has seven parameters, as shown in the table below, alongside the values used for each of four baseline AI 'characters'.

Parameter	Description	Speedy	Shadow	Bashful	Pokey
ENEMY	Plus for each dead enemy	0.4	0.4	0.3	0.6
TEAM	Plus for each living ally	0.1	0.2	0.3	0.0
WOOD	Plus for each destroyed wood	0.2	0.1	0.0	0.1
KICK	Plus for a Kick power-up	0.3	0.15	0.0	0.15
BLAST	Plus per Blast power-up	0.15	0.15	0.15	0.15
ALLY_DISTANCE	Plus per square away from ally	0.00	-0.01	0.02	0.01
AMMO	Plus per Extra Bomb	0.0	0.05	0.1	0.0

Negotiators

When creating AI negotiators, it was important that each negotiator was varied. So two things were implemented to achieve this: a heuristic value and a random element.

Heuristics

The heuristics portion of negotiators is simple. A weight is attributed to the different powers an agent can have (amount of bombs, blast strength and kick power). They are multiplied by the power they are attribute to find a heuristic value. The weights are currently set to be between 0 and 1, however, they can take any value.
These weights are then used to make decisions on certain agreements.

- Agreements dealing with bombs use the blast strength and amount of bombs.
- The no kick agreement uses the weight for the kick power.
- Additive agreements (alliance and share vision) use all three.

There are two types of heuristic negotiators: basic and random. Basic heuristic negotiators will always propose agreements with the highest heuristic values, and accept agreements if their heuristic value is above 0.5.
Random heuristic negotiators will perform similarly, but will always create a random number, between 0 and 1, to compare the heuristic to. If the heuristic is bigger than the random number than the proposal is made/accepted. This ensures some variety between games, even if the players are always evenly matched. For this reason, only the random heuristic negotiators are used in the released version.

Personalities

To set the personalities of each agent the weights were set as such:

Parameter	Description	Speedy	Shadow	Bashful	Pokey
Blast Strength	Plus for each range of bomb blast	0.75	0.5	0.1	0.5
Ammo	Plus for each each bomb an agent can place	0.2	1.0	0.6	0.2
Kick	Plus if an agent can kick	0.2	0.1	0.1	0.2

This ensures some agents are more likely to do certain types of agreement than others. In fact, Bashful is the least likely to make or accept an agreement in general and Shadow is the most likely to do it. However, overall all agents should make agreements frequently, as that is the change to the game that was implemented, and we wanted to highlight that.

Implementation/Instructions

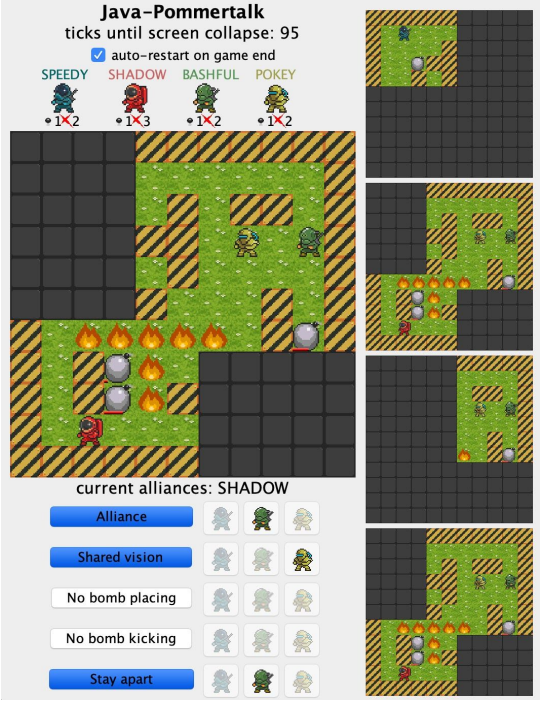
To run an implementation of PommerTalk run one of the .bat files. “Run Pommer talk AIonly.bat” will run PommerTalk with 4 AI players. You can swap between the different views using keys 1 through 4, or see the overall map with key 0. “Run PommerTalk Human.bat” will run the game with Speedy as a human player that you can control. Once the human player dies the game is treated as an AI only game and views can be swapped until the game ends.
To learn how to play the game please read : PommerTalk Tutorial.

Future Work

Negotiator agents are simple versions of the ideal agents. At the moment there is a high amount of randomness, skewed towards ideal agreements. This creates two small issues: lack of agreements at the start and skewed towards certain types of agreements. Since making and accepting agreements is tied to the power of each agent, at the beginning of the game, when agents are weaker, agreements are less likely. Also, the heuristic functions are not ideal for human interaction. While they work well for AI only games, where agents tend to perform optimally, against a non optimal human, agreements with humans will often be declined.

Perhaps the best way to improve the negotiators would be to increase the amount of variables they keep track on. This would allow for more complex and interesting heuristics and give them more distinct personalities. Keeping track of how many times another agent placed a bomb nearby, or stepped away from another agent, could influence negotiations in interesting ways. For example, a shy agent may be more inclined to make deals with other agents which keep away from it.

Another improvement would be simply keep better track of the current variables individually. At the moment each agent interrogates the game state to find the power of other agents. This means some variables are not reliable (such as ammo which changes depending on how many bombs are placed) and other variables are not nuanced enough (kicking is simply true or false). So while the heuristics work well, there are several improvements possible in the future.



Video

A video of the game in action showing gameplay with a human player and as an observer watching four ai agents playing, can be seen at the following link:
https://youtu.be/KPBOuO0dyK0