Junior Honours project report

Keanan Frazer, Bethany Gillam, Liam McMenemie, Christa-Awa Köllen, Jack Neilan
University of St Andrews

Submitted 2017-05-18

Abstract

The abstract should be about 100 words long. Quisque consectetuer. In suscipit mauris a dolor pellentesque consectetuer. Mauris convallis neque non erat. In lacinia. Pellentesque leo eros, sagittis quis, fermentum quis, tincidunt ut, sapien. Maecenas sem. Curabitur eros odio, interdum eu, feugiat eu, porta ac, nisl. Curabitur nunc. Etiam fermentum convallis velit. Pellentesque laoreet lacus. Quisque sed elit. Nam quis tellus. Aliquam tellus arcu, adipiscing non, tincidunt eleifend, adipiscing quis, augue. Vivamus elementum placerat enim. Suspendisse ut tortor. Integer faucibus adipiscing felis. Aenean consectetuer mattis lectus. Morbi malesuada faucibus dolor. Nam lacus. Etiam arcu libero, malesuada vitae, aliquam vitae, blandit tristique, nisl.

Declaration

Authors were members of project group G.

Contents

Introduction

Describing the problem the team set out to solve and the extent of its success in solving it. The section should outline key aspects of the project for the reader to look for in the rest of the report.

This practical was meant to challenge one's ability to effectively work in a group, for the to organise tasks amongst themselves, interface with other teams to develop a standard protocol compatible with all implementations, set and adhere to self-imposed deadlines, and finally to deliver a product by the final date. The task was to build Settlers of Catan, a strategy board game, and have it equipped with AIs, the ability to play games over the network, the ability to network with other teams and to of course have a working implementation of Catan's basic rules and game logic. All of this was to be accomplished at the discretion of the group, through experimentation and refinement where necessary, as well as through the successful delineation of tasks amongst the group's constituents.

This implementation achieves all of the basic requirements, including the ability to play with AIs (either locally or remotely), the ability to play as an AI or a player, the ability to play an off-line game, the ability to play as a player networked into a remote server, and the ability to network with other groups' implementations. The UI and front-end utilise libGDX for rendering of art and graphics, and the server is exposed through the abstraction of the actual implementation of a Client's connection. For example, the server is able to be run with remote players through Java's standard TCP sockets, or alternatively through a local connection all via the same code and turn flow; in the case of it being locally run from a client, the server is capable of being run on a different thread (spun off of a 'LocalClient' or 'LocalAIClient') complete with complex, concurrent data structures and semaphores to ensure the correct behaviour. Conversely, the server is capable of being run as a stand-alone program, where clients can simply connect via TCP sockets. Given that connections are abstracted over, these features can all be mixed and matched, depending which components are plugged in with the Server (e.g. a completely offline game with four AIs, an offline game with one human player and three AIs, or an off-line game with a human player, and any combination of remote clients and local AIs to fill in the blanks. Java is used throughout the entire project, for the coding of the game logic, the server, and the client.

Project details

Summarising the achievements of the project, and discussing those aspects of the project that are of particular interest: the main ideas of the design, unusual design features, special algorithms and data structures, odd implementation decisions, novel user interface features, etc.

Protocol design

Early on, it was decided that the intergroup protocol would be implemented via Google's Protocol Buffers, as opposed to JSON. This is due to the inherent advantages protocol buffers entail, including much more compressed message size, and an extra level of validation and definition in what the protocol messages entailed and had to be. This is in contrast to JSON, where ultimately someone could put a string in the place of an integer, meaning there is a substantially higher chance for error, and therefore a need for a large amount of message validation and sanity-testing.

The protocol structure is as follows: the overarching message type is a 'Message.' A 'Message' has a 'one-of' field, which simply means that a message can be either a 'Request' or an 'Event.' These message types represent communication from the client to the server, and the server to the client, respectively. An event can be sent either just to the sender, in the case of secret information or an error occurring in the previous request, for example, or can be sent to every player as a receipt of a valid move that was carried out. An example of secret information would be the exact contents of a card which was stolen, or which development card ended up being bought. In these cases, the original event is sent to the player in question (if a steal, this could be sent to both players), and then the relevant information is obscured before being sent to other players. For example, in the case of stealing a resource, the resource type is merely altered to be 'Resource.Kind.Generic'.

A 'Request' has a one-of field for each defined request type (including building a settlement, road, or city, playing a development card, buying a development card, starting a trade, choosing a player to steal from, choosing a resource, etc). An 'Event' is similar in that it has a 'one-of' field, meaning it can take the form of any number of 'Event' types which roughly correspond to the 'Request' types (including settlement built, city built, road built, resource stolen, development card bought, development card played, trade occurred, etc.). An 'Event' also has an 'instigator' field which simply entails the 'ID' of the player who sent it. While this is almost always the current player, including this information was important for debugging purposes.

This implementation is coded fundamentally in terms of this group protocol, as it involved the simplest approach, and less boilerplate code for validation and message translation.

Board design

The board is the root of what "Settler of Catan" is, and as such, coding and representing the board was worked on first. The HexGrid entails a Hash map of 'Point' objects to 'Hex' objects, a hash map of 'Point' objects to 'Node' objects, a list of 'Edge' objects, and a list of 'Port' objects, a specialisation of 'Edge.' It also maintains which hex has the robber. To construct the hash maps and data structures in this manner is expensive in terms of time, but results in exceptionally fast retrieval. This was deemed to be a fair trade-off, as retrieval of objects will occur infinitely many more times than the initial set-up. Due to this, it would present itself as a notable inefficiency in the data structure to have lists as opposed to hash maps. Likewise, the board is generated long before anyone even requests it anyway, making the slightly longer set-up time negligible.

The 'Point' class was used as the key type for the aforementioned hash maps due to one particular quality: the Point class is hashed purely based on its 'x' and 'y' integer fields. In this way, hashCode(new Point(1,1)) == hashCode(new Point(1,1)). This quality makes 'Point' act a bit like a tuple in this instance, which made it an excellent choice to represent the coordinates of the board, and to be keys for the data structures maintaining it; for example when coordinates are received over the network as a part of a certain move, say a 'BUILDCITY' move, then the node can easily be retrieved through extracting the x and y values, and doing: grid.getNode(x, y). Then further checks can be performed and the move can be validated and carried out. Another important note of the design of the board involves the abstraction over elements that go into a HexGrid. To start with, a 'BoardElement' is used to describe anything on a board, which includes an 'Edge' or a 'GridElement.' A 'GridElement' is used to define anything on the grid that can be defined with x and y coordinates ('Hex' and 'Node'). These abstractions are leveraged in forming a list of potential, valid objects that can be built upon (reference MoveProcessor.java), for example, and in storage of objects in a single data structure (since they all implement BoardElement). This is utilised when generating valid moves based on the game state, which is used in the UI as well as in the AI's move decision process.

Despite the data structures for the board making sense, populating them with values became a lot trickier. The coordinate system used in the group protocol ended up being the most appropriate for our implementation, as it defines Nodes and Hexes on one coordinate system, allowing for less coordinate translation and object manipulation all around. This is due to the way in which the x and y axes are defined relative to the hex grid; as such, the board, its boundaries and which coordinates represent a 'Node' vs a 'Hex' are all formalised mathematically. In this way, one can set up nodes and hexes through two nested for loops, systematically iterating from one end of the coordinate system to the other. Using a simple check (x % 3 == 0 indicates a 'Hex'), the various nodes and hexes can be created easily. If the coordinate represented a 'Hex', then a chit and a resource are randomly assigned based upon a statically-created map of remaining resources and chits. From there however, linking up the nodes and hexes in their complex web of relationships was a lot trickier. For this, all neighbours of a 'GridElement' are retrieved. A 'neighbour' of a 'GridElement' entails all other 'GridElement' objects which have a distance between 1 and -1 in their x coordinates, and the same with their y coordinates (with the exception of both differences being equal). This results in all 6 neighbours: for a hex, it retrieves 6 nodes, and for a node it retrieves between 1 and 3 hexes and 2 or 3 nodes (depending upon board location). In having the list, then mutual references can be established, allowing for simpler algorithms down the road (e.g. retrieving the resource type from a hex adjacent to a settlement (node), useful for resource allocation); the neighbours are simply iterated through, forming all appropriate relationships between objects on the board. If the neighbour is a node, then an edge is created (if not already created previously) as well.

With nodes and hexes having their references established and 'Edge' objects having been created, 'Port' objects needed to then be created. Ports can be ascertained by the mathematical boundaries of the board (i.e. check if one of the 'Edge' object's nodes is on the boundary), and then checked to see if any other port is within a certain distance of that location. If so, then the location is invalid. This allows for ports to be randomly, uniformly distributed across the board. To detect if the given node is too close to another port, known ports are iterated through and their distance is checked. This is done through an algorithm which navigates along edges from one node to another, which is guaranteed to find a path which may or may not be the shortest. This is done by comparing the overall distance between the current node and the goal node, and both adjacent nodes and the goal node. This allows us to ascertain which adjacent node gets us closer to the goal. This whole process gives an upper bound on the shortest distance between two nodes, and is used as a reasonable measure to separate ports uniformly. It fails in finding the absolute, shortest path between two nodes, when two nodes at any point in the journey happen to share this distance heuristic. In this case, the path taken may take the longer path around a given 'Hex.' Even so, the algorithm provides a reasonable measure to check against.

Game design

A 'Game' is a raw game-state class used to define the information common to the representation of state on both the client and server sides. The idea of a game is modelled using inheritance; Game is the core representation of a game (including a board, basic information about players, fundamental methods, etc.), while 'ServerGame' is a specialisation which includes other server-specific information and methods that go into running the game (i.e. resource allocation, changing turns, joining games, etc.), and 'ClientGame' is a specialisation which includes client-specific information and methods. 'Game' and all of its specialisations simply add the capabilities to manipulate the underlying board (a 'HexGrid' object), and information held in instances of 'Player.' In this way, they add the notion of "state" on top of the core representation of "Settlers of Catan." This approach allowed for the most amount of code re-use possible, since the client and server need to have the same underlying notion of a game state anyway. In this way, 'Game' also entails a lot of common functionality that both the server and client need, including methods to check if a road was broken, to update largest army, or add a new player, for example. The 'Game' also entails a grid and basic structures for maintaining players, the bank, ids, etc.

The primary difference between these two specialisations of 'Game' is the way in which they interact with the larger game flow; that is, 'ServerGame' is essentially 'Game' with additional logic to process requests, with 'ClientGame' being similar except with logic for processing events. Likewise, a 'ServerGame' acts as the authoritative view for the entire instance of "Settlers of Catan," across all players, while 'ClientGame' acts a 'view' on this overall state through the perspective of a Client. 'ServerGame' essentially entails a series of functions which correspond to each 'request' type defined in the intergroup protocol. These methods all entail extensive exception usage which, when thrown, indicate to the server's MessageProcessor (which called this method) that the request failed. If so, then the exception is caught, and an error event is subsequently set up and sent back to the corresponding client. This exception handling allows for rules to be iterated upon / altered in a very easy, transparent way. Likewise, it provides a uniform, understandable way of indicating when something goes wrong. With all errors going down the same code path, regardless of origin, there is far less code re-use and less boiler plate code for checking if methods return 'null' instead of a proper value, etc. This system also involves incredibly detailed error messages which can be sent back to the 'Client' in question. Some examples of exceptions being thrown are if a user cannot afford something, if the bank cannot afford something, and if an object already exists where you are trying to build. Other, similar edge cases being covered in this way as well. This is in contrast to 'ClientGame,' which entails a series of functions which correspond to each 'event' type that is defined in the intergroup protocol. These methods process each event type and alter the given game state and / or player data, from the perception of the 'Client' at hand. This has similar exception handling, except to a lesser extent given that the server has to be trusted as having the authoritative view.

A few pieces of functionality that are common to 'Game' involve all the functions for longest road detection, the function for largest army detection, as well as the function for ascertaining a player's new resource allotment based upon a given dice roll. The functions for longest road cover the different cases in which road lengths change based upon road location, including connection a chain, building on the end of a chain, having a new foreign settlement break a chain, or having an existing foreign settlement break a road connecting another. These functions all get invoked when a new road is built. As is in the case of a foreign settlement breaking a road, its method is invoked whenever a settlement is built instead. These methods all rely on how roads are stored in the Player class; they're stored in lists of lists with each list representing a single autonomous chain of roads. This data structure, while involving expensive insert operations, leads to far simpler detection of longest road which happens with a lot more frequency. As such, the operation of checking longest road is far simpler in concept and complexity (just check the length of each road chain), which is a fair trade off.

Due to a player's roads being stored as lists of lists, the methods for longest road therefore operate on these individual lists whenever a given road or settlement is built, and decide whether or not two lists need to merged, whether a settlement's node is present in the list of edges and then also how to break up that chain into two, which chain to insert an edge into, or whether the road represents an entirely new chain. To check if an edge belongs in a given chain, all edges of each list are checked to see if one is directly connected to the new road (if they share a node). If so, then the road is added to that chain. If however the road is added to two chains, then all the elements of one list are added to the other, the former list is removed entirely, and with the new, duplicate road being eliminated in the joined chain to keep everything consistent. For a settlement breaking a road, the edges associated with that node are checked to see if they have a road of another colour. If so, then that player's road chains are checked for a split.

With all this logic in place, whenever a settlement or road is built, a function gets called to check for longest road. It performs any revoking and granting of the victory points (VP) associated with the longest road card. The largest army detection logic is similar in that it gets called anytime a player plays a knight development card. This function checks the size of the player's army who currently has largest army, if any, and performs any revoking and grant of the associated victory points if necessary. Allocating a resource was the last, mentioned example of functionality common between 'ServerGame' and 'ClientGame' that is therefore present in 'Game.' This function takes a player colour and a dice roll, and checks all the player's settlements to see if any is present next to a hex with a chit matching the dice roll. If so, then 1 or 2 resources are added to a map of resources to integers to be returned, depending on if the node has a settlement or city, respectively. This is repeated for all settlements, and the map can then be returned. Once all players have the resources they should be allotted for the given turn, then the maps for each player can be checked for inconsistency. For example, if during the initial process it was discovered that a given resource cannot be allotted, then this resource is eliminated from EVERY players' map of new resources. Once this sanitisation process has completed, then the resources can be granted to each user, with the maps being converted and sent out as part of the 'ROLLED' event.

Similarly to 'Game,' 'ServerGame,' and 'ClientGame,' 'Player,' 'ServerPlayer,' and 'ClientPlayer' have an essentially comparable relationship dynamic in terms of how logic is allocated; as one would guess, 'Player' contains all logic for manipulating information about players that is common to both the server and the clients, while ‘ServerPlayer’ and ‘ClientPlayer’ contain special logic that only applies to either circumstance. For example, as ‘ServerGame’ performs a multitude of checks to see if a request lines up the given game state, ‘ServerPlayer’ entails a lot of the functionality for checking if a request lines up with the given player state. This represents a layer of functionality in the backend, as well (to be discussed further in “Server Design”). Contrastingly, ‘LocalPlayer’ doesn’t entail much logic at all, aside from a few minor things which slightly differ from ServerPlayer’s version. This abstraction allows for a lot of code re-use as well, as common functionality is shared in ‘Player.’

Server design

The backend was designed using a modular approach, with very clear layers of functionality. The server is the top layer, which coordinates communication between all threads listening to players. It retrieves messages from each ‘ListenerThread,’ and passes these messages into a ‘ConcurrentLinkedQueue’ in the subsequent layer, the ‘MessageProcessor.’ The ‘MessageProcessor’ is the part of the server which reads messages one at a time, breaks them down, and parses them for accuracy and validity based upon the current game state. It keeps track of which moves have been seen and if any players have any moves which they are expected to make, and uses this information to help deduce if the received move is valid. If so, it also determines what action to take and then performs it. Often this includes going one layer deeper in the server into the ‘ServerGame,’ which acts as the authoritative state for the overall “Settlers of Catan” instance. The protocol-buffers-compatible representation of the request from ‘MessageProcessor’ is transformed into one understandable by the internal game logic, and then the action is carried out if all internal information is valid.

Parsing of the protobuf class ‘Message’ is quite easy, thanks to the extra mechanisms protocol buffers entail. When a message can be ‘one of’ something, an enum is implicitly set in the protobuf object which tells you which option was set, allowing for easy retrieval. This makes parsing requests very simple, as all one needs to do is ‘switch’ on the field representing what field of the ‘Request’ is set. This tells you exactly what information to extract from the request, and in most cases then calls the appropriate method in ‘ServerGame’ to handle that request type. In the cases of chat messages or trade requests to other players, these are instead just forwarded to the remaining players. Otherwise, the internal request is sent to the ‘ServerGame’ for further processing. If the processing succeeds with no exceptions or other errors, the given request is run through a method that determines what expected moves are now expected from each individual player. These expected moves are used in determining if an incoming request is valid as well, along with other basic information like which player sent the request, whose turn it is, and similar information. Before being further parsed, the extracted request is run through a series of checks like this. Due to expected moves being so fundamental to determining if a move is valid, every request which succeeds needs to be ran through a method to eliminate the successful move if it was already expected from that player, as well as determine if any players have new expected moves. For example, if a 7 is rolled, then any player with over 7 resources needs to send a ‘DISCARDRESOURCES’ request, and the player who rolled needs to also send a ‘MOVEROBBER’ request before the game can proceed.

In thinking about the way the server should communicate with the clients, it was noted that it would have to be general enough to be able to handle a local or remote client. This would allow AI's to be able to be run directly from the server-side as well, and for a client to be able to host a server locally. Likewise, regardless of the type of client, the Server needs to be able to be constantly listening for messages from all clients, as in some instances messages are required when it isn't the player's turn (I.e responding to a trade, discarding cards, etc). Due to these needs, a class 'ListenerThread' was created which unsurprisingly, runs in its own thread. This addresses the need of being able to constantly listen for messages, as each thread simply needs to loop listening for incoming messages from the client. When found, it adds this new message to the 'ConcurrentLinkedQueue' in the Server's 'MessageProcessor,' so that the message will be handled once it's at the front. The concurrent nature of the queue allows for each individual 'ListenerThread' to populate the queue without needing to worry about acquiring a lock first, given that information could be lost or corrupted if this race condition were not handled. Likewise, it means the Server's 'MessageProcessor' doesn't need to worry about this either, as it simply needs to 'poll' messages from the queue and process them one at a time.

To address the first need of having local or remote clients, the 'ListenerThread' sends and receives its messages through an 'IClientConnection' object, representing either a local OR remote client, as opposed to having 'ListenerThread' implemented strictly in terms of a TCP connection. This abstraction is necessary as it means the ListenerThread needn't care of the underlying implementation of its 'IClientConnection.' The two classes which implement this interface include 'LocalClientConnection' and 'RemoteClientConnection,' and simply entail functions for sending and receiving messages from the corresponding 'Client.' The latter is simply a wrapper over a TCP socket, which allows the Server to be connected with remote clients, regardless of implementation (provided they adhere to the group protocol), while the former also has an object reference to a 'LocalServerConnection' object, which is the analogous version for a 'LocalClient' object (to be discussed later), thereby creating a bridge between the local client and the server (run in separate threads). This is the mechanism which allows for local AI's to be run from the Server, or for a local Server to be run directly from a client, the combination of which would allow for an "offline mode." In summary, 'ListenerThread' fulfils the aforementioned requirements, in allowing for all users to be listened to constantly (so no messages are lost or delayed), and for the sending and receiving messages regardless of the context of the connection. This means the same code can be used for having local OR networked clients, as with the correct abstraction mechanisms, the nature of the client doesn't impact the flow of communication whatsoever – only HOW the communication is sent.

Client design

Given the way the 'ListenerThread' objects on the Server abstract over the type of connection, a Client needed to be abstracted over to indicate what components were necessary for ALL clients, regardless of implementation to be plugged into the server (either Remotely or Locally). These core components of a 'Client,' inherent to a 'Client' if they wish to hook up to our server, include an 'EventProcessor' (continually listens to the server, either locally or remotely, and processes incoming events while updating the state and expected moves), a 'MoveProcessor' (generating a list of ALL valid moves for a given turn, and if a given request is valid), a 'TurnProcessor' (constructing a request message and sending it off the server, either locally or remotely), and a 'ClientGame' (representing the overall state of the underlying game, from the Client's perspective and to the best of its ability) all of which share a reference to the corresponding 'Client.' These core components are defined and are fields in an abstract 'Client' class, given their functionality is constant regardless of the type of client (AI vs Player, and local vs remote); this means that any actual implementation of a client simply needs to extend the abstract 'Client' class, instantiate its fields properly, and add any additional modules or information to maintain. This is the most modular approach possible, and means one only needs to write client logic ONCE, and can then make local and remote specialisations which override two methods (setUp and shutDown) so that it can plug into the existing code. For example, 'AIClient' extends Client, entails all AI logic, and is then extended by 'LocalAIClient,' 'LocalAIClientOnServer,' and 'RemoteAIClient,' which simply have involve different 'IServerConnection' and 'IClientConnection' objects. These different connections are of course reflected in 'EventProcessor,' 'TurnProcessor,' and 'ListenerThread'). With those core, three relationships properly set up between client and server, no other code needs to be changed to make a local client available remotely, and vice versa.

Due to all the interacting aspects of an abstract 'Client,' ensuring thread-safe, consistent communication between the different components was essential. This becomes even more necessary when thinking about a client in relation to a graphical user interface (GUI), which would need to run in its own thread as well for the sake of responsiveness. For the core 'Client,' mechanisms needed to be introduced to ensure that as the 'EventProcessor' received events, that the associated 'ClientGame' be updated in coordination with the GUI and AI, in order to facilitate the remaining in sync of all modules with one another, and to ensure no corruption of data or other race conditions. Likewise, these mechanisms needed to be general enough so that they would function in the opposite way, as well as would accommodate future additions to the code easily. To implement these requirements, it was noted that a notion of a 'lock' would be needed whenever accessing or updating the 'ClientGame,' so that other threads would be forced to block until the resource became available. If the thread has the resource however, then it can conduct its business and subsequently release the lock, thereby allowing another thread to resume. For example, if the GUI is wishing to update the board, it first attempts to acquire the relevant lock from its 'Client' object, and then it can ensure that the view of the state it is getting is up-to-date and current. In order for this mechanism to be employed correctly, all participating threads that go into a client need to be spinning and waiting for the given lock at all times, if it doesn't already have it. Due to this and for reasons discussed in the subsequent paragraph, the entire 'Client' class implements 'Runnable' (so that it may be run in its own thread). Its main 'run()' method is simply an infinite while loop which attempts to acquire the lock associated with the client's 'ClientGame' and the one associated with the Client's 'Turn' (to be discussed later), and then reads an event if available, followed by updating the state. Other threads which desire the 'ClientGame' then need to acquire the lock, which then essentially interrupts this thread, as it will then need to block until it can receive the lock resource once more.

'EventProcessor' is intended to operate in its own thread, and has a reference to an instance of an IServerConnection. This connection, to be discussed more thoroughly later, abstracts over the nature of the connection with the server, and simply facilitates the sending and receiving of messages to and from the server, respectively. ‘EventProcessor’ only focuses on the receiving of messages; it handles the parsing of received messages and events, and then passes them off to the corresponding Client’s ‘ClientGame,’ to handle the processing of the new information about the game. The ‘EventProcessor’ needs to run its own thread as the server sends events inconsistently, unexpectedly and from all players; as such, the client needs to be constantly listening to the server so that no messages are missed. When the event processor is processing a message and stripping out the internal ‘Event’ (see “Protocol Design”), or rejecting it if it is not there, it determines the type of the event in a ‘switch’ statement, and therefore also which method in ‘ClientGame’ to call to process it further. It also runs the event through a method which analyses the event, the instigator, and any other participants to determine any new expected moves for this player (i.e. a discard request when seeing a dice roll of ‘7’ while having more than 7 resources, or a response to a trade when one is received, etc.). There are some simple checks in ‘EventProcessor,’ but it is assumed that the server has an authoritative knowledge of the state of the game meaning that these are just a formality. This is justified by the fact that if something does go wrong due to the server, the client will be unable to recover anyway. o

Before understanding how moves are sent to the server, it is important to note what a ‘Turn’ object is. The ‘TurnProcessor’ works by processing the main ‘Turn’ object in ‘Client.’ This object is quite important, as it entails all information about the current turn’s state, including dice roll, expected moves, whether-or-not it is trade phase, as well as the current information about the desired request the user wishes to make. Given the information it entails, it is no surprise that this object is also utilised and altered by the different components of a client; as such, it necessitates the use of another lock as well. The ‘Turn’ object is the primary way for a generic ‘Client’ to set up moves, regardless of how it provides moves to the ‘TurnProcessor’ (AI, Player, etc.). Essentially, the ‘Turn’ object entails a field for ‘ChosenMove’ which is an instance of the enum type used in a ‘Request’ (see Protocol Details), as well as fields which describe further information about all possible request types, with relevant pieces of information filled in. For example, a ‘Turn’ object representing a ‘BUILDSETTLEMENT’ move would have the ‘chosenMove’ field set to ‘BUILDSETTLEMENT,’ as well as having the ‘chosenNode’ field filled in as well. This object extends to all requests in a similar way, meaning it therefore is capable of encapsulating all possible information about a desired move. This object type is therefore used in multiple scenarios, such as generating moves (MoveProcessor.java), as well as in forming potential moves and updating the main ‘Turn’ object so a move can be processed and sent.

‘TurnProcessor’ operates with the same ‘IServerConnection’ that the Client’s ‘EventProcessor’ has, but conversely only focuses on the sending of moves to the server. In having all communication with the server abstracted over with one object, the ‘IServerConnection’ is the only element on the client-side that is needed to separate a remote connection from a local one. A ‘RemoteServerConnection’ is essentially just a wrapper over a java TCP socket, while a ‘LocalServerConnection’ simply just has a reference to the Server’s corresponding ListenerThread’s ‘LocalClientConnection.’ This ‘bridge’ is facilitated through two ‘ConcurrentLinkedQueues,’ one of which is in ‘LocalServerConnection’ and the other of which is in ‘LocalClientConnection.’ These queues represent a stream of messages being received from the other end of the connection, and in this way, messages can easily be passed between a generic, local client and server (AI, Player, etc). 'TurnProcessor' as a class focuses on forming the request to be sent to the server, and then it calls a method to validate the request from the client's associated 'MoveProcessor.' It forms a request by, provided it has the lock, looking at the Client's 'Turn' object, 'switching' on the 'chosenMove' field, and then extracting out the relevant information. These two bits are used to form a 'Request' object (see "Protocol Design"), which is then sent to the associated 'MoveProcessor' to be validated against game state and expected moves. If the request is valid, it can then be sent to the server via the Client's 'IServerConnection.'

The 'MoveProcessor' class is tasked with validating moves based upon the current game state. The game state it has is the 'ClientGame' object that the associated 'Client' instance has. This class has a method for each unique type of request that dictates if a given move would be valid, based upon the current game state. These methods entail checks such as if it's the player's turn, if they can afford the thing they are about to buy / build, if the move is expected, and similar logic specific to each request. For checking a given request based off of known expected moves, the list of expected moves is first checked to be empty. As briefly mentioned before, the expected moves are retrieved and updated via the cross-thread 'Turn' object, associated with the given 'Client.' If the expected moves are empty, then remaining checks for the given move can be carried out. If however, the list of expected moves is NOT empty, then if the given request type is NOT present in the list of expected moves, the move is invalid. If the request is in the list, then additional checks for the request may be carried out. Here, expected moves are NOT eliminated from the list as this is done with the client's 'EventProcessor;' since events verify whether a move was indeed carried out, then this is the only place where expected moves can be added or removed based on principle. This is necessary due to the fact that some moves require multiple elements to be fulfilled. For example, playing a monopoly development card involves first sending a move to play a monopoly development card. If this succeeds, then the player must then send a 'choose resource' request. Likewise, playing a 'Year of Plenty' development card involves two subsequent 'choose resource' requests, and moving a robber entails moving the robber, then submitting a player to steal from. If the client were to not maintain when moves are expected, AI's would need to have hard-coded elements to remember when they had to play a certain move after another. Having the expected moves integrated into a client's fundamental move and event processing modules makes move validation for any client infinitely more seamless.

'MoveProcessor' also entails functionality for generating all valid move possibilities, as this is used in some way for all sensible, generic implementations of a 'Client.' For example, an AI could use this functionality so it doesn't have to generate moves itself, and likewise a GUI could use it to display valid options and give visual cues to the user. This functionality works by systematically checking every move possibility, and running the corresponding method in 'MoveProcessor' for checking the validity of that request type. Using these functions as a filter, all move possibilities can easily be filtered. For example, an internal function in 'MoveProcessor' called 'genBuildingPossiblities' returns a list of all BoardElements ('Node' objects, 'Edge' objects, and 'Hex objects') which can be built upon. This method looks at all nodes and edges in the Client's 'HexGrid' (stored in 'ClientGame'), retrieves the local player for this 'Client,' and checks if the 'Player' can build on the given node or edge, or upgrade the given settlement. This is performed through the corresponding functions in ‘MoveProcessor,’ which are implemented in terms of analogous functions in the abstract ‘Player’ class. These are common to both a ‘ServerPlayer’ and ‘LocalPlayer;’ that is, it simply checks if the given player can build there. If so, a ‘Turn’ object is constructed using the ‘BoardElement,’ and added to a list. With all building possibilities sorted, then each remaining function in ‘MoveProcessor’ is called with each possible valid input. If the function returns ‘true,’ indicating that the move option is valid, then ‘Turn’ objects are constructed with the given request type and additional info. This forms a master list of valid moves, and is subsequently returned.

AI structure

To begin with coding the AI, the generalisation of what is means to be a client needed to first be fulfilled. With this, then the AI could be implemented through little additional code, beyond the AI logic, of course. This is due to the way in which AIs are able to hook into the Client framework. Essentially all an ‘AIClient’ needs to do is to extend ‘Client,’ override Client’s run() loop, and implement some AI which has a method of retrieving all valid move possibilities. With this, then each one simply needs to be ranked in some way, with the one with the highest rank chosen and sent via the client’s ‘TurnProcessor.’ Luckily, ‘MoveProcessor’ is an aspect of every ‘Client’ implementation, and thus any AI simply needs to do for getting move possibilities is to call on ‘MoveProcessor,’ as it already knows how to retrieve the list of valid moves for the given state of the turn. This, in turn, means that an AI doesn’t need to know or care about which moves are expected; in the case of a move being expected, the returned list from ‘MoveProcessor’ would merely only contain moves of that type, provided there are no other constraints present. This means an AI simply needs to go through the stream of valid move possibilities and assign a rank, vastly simplifying its job and reducing the amount of code redundancy. If the AI had this functionality from ‘MoveProcessor’ directly, then it would essentially be causing the processing and recording of this duplicate information. The current approach is vastly more efficient and modular.

In order to keep the ‘AIClient’ as simple as possible, it was made abstract similarly to ‘Client,’ as well as given an abstract object as a field, ‘AICore,’ which entails all processing and decision-making functionality. The only thing this leaves for ‘AIClient’ is to have its own run() method; its version of run() differs from that of the one in ‘Client,’ as this one needs to incorporate logic for making moves as well as processing events. The loop structure remains entirely identical, however, in that the loop tries to acquire locks before performing a given action. The run() method for ‘AIClient’ is an infinite while loop (for all intents as purposes), which acquires locks, tries to process an event, releases locks, sleeps, acquires locks, calls an abstract method for generating a move from the ‘AICore’ if given conditions are met (it is the AIs turn OR a move is expected from the AI (i.e. discard request)), and then releases its locks again. Alternatively an ‘AIClient’ could’ve used the same run() method as ‘Client,’ if the ‘AICore’ implementation ran in its own thread as well, however this presented an issue in terms of synchronisation; for example, this meant that an AI couldn’t move onto another move until it received some sort of feedback in the form of an event. This was attempted, and simply involved the ‘AIClient’ hanging and having a lot more downtime. In this case, the AIClient’s behaviour with locks, making moves and processing events essentially amounted to synchronous behaviour anyway. Likewise, it was a lot more complex of a program, with one extra thread for each ‘AIClient,’ and harder to debug and ensure thread safety. The current manifestation of AIClient's run() method produces no noticeable downsides, and is worth the trade off. Likewise, it is all that needs done to ensure that the AI can hook up with the server.

In order to actually implement the AI's logic, an interface 'IAI' was written to describe the methods needed by an AI in order to properly hook up with an 'AIClient.' The main one is the performMove() method which coordinates the relationship between all the remaining abstract functions defined in the interface. These include getMoves(), rankMoves(), selectMove() and selectAndPrepareMove(), while the remaining functions defined in 'IAI' are to be implemented in the actual implementations of 'AICore.' This approach was decided upon as it involves the most amount of code re-use as possible; this is because any AI logic implementation will need a method of retrieving valid moves, ranking them, selecting them, and preparing the move for being sent off to the AIClient's associated 'TurnProcessor.' Ranking moves is the bit in which the actual AI logic comes into play, and could involve anything from simply assigning numbers, to a full-blown Monte Carlo Tree Search or a something similar. As of now, AICore's implementation of rankMoves() simply switches on the 'chosenMove' field of the given 'Turn' (from the overall list of possibilities), and then calls the appropriate abstract heuristic function. AICore's current implementation of selectMove() simply takes a list of all optimal moves (that is, all 'Turn' objects which share the highest ranking), and randomly chooses one. With the current structure, all one needs to do to write an AI that can interact with a server is to create a class which extends from 'AICore,' and simply implement the given functions for heuristic calculation. 'AICore,' as described above, already handles the manner in which these components interact. In the described manner of architecting AIs, implementing a random AI simply entails overriding every heuristic function to return the same number (i.e. 0). In this way, every possible move will be a part of the 'optimalMoves' list parameter to AICore.selectMove() (that was generated from rankMoves()), and the function will simply choose a random one.

Given that 'AIClient' can have any implementation of 'AICore' as its underlying AI logic processing unit, the described system for AI implementation is flexible, easily extensible to future heuristics, and conducive to iteration and improvement. Likewise, this also means that 'AIClient' and its children classes are the only classes needed to hook up ANY implementation of 'AICore' to the server for playing a game. 'AIClient' has three classes which inherit from it: 'RemoteAIClient,' 'LocalAIClient,' and 'LocalAIClientOnServer.' The latter entails an 'AIClient' which operates directly from the server, operating on a different thread, while the former two represent an 'AIClient' running with a GUI, remotely and locally from the server, respectively. The former two are analogous to 'RemoteClient' and 'LocalClient,' respectively, and essentially entail the exact code as the corresponding class (except for which class it inherits from, and a few other things).

Intergorup protocol and the process of integration

The intergroup protocol design became pivotal to how this implementation was written, organised and structured. A group representative, 140001596, attended nearly every intergroup meeting (aside from once or twice when external obligations came into play) and worked heavily to write the protocol and provide alternative approaches and perspectives (advancing this group's stakes in the matters at hand). For example, in order to begin coding over Christmas holiday, this group member wrote a protocol using Protocol Buffers for our group to work off of. This was based on knowledge of what the group protocol would need to entail, but at this juncture, the intergroup protocol was too minimal to provide anything to go off of. This led to the foundation of our code base, backend, and server, and the original plan was to map our internal protocol to whichever one the groups eventually decided upon. This proved to not work as well as planned, as far messier code was written than had been intended. Likewise, it involved an entirely new module to do the actual translation. Due to this, our group representative began being more active in group meetings, and worked to bridge the gaps in protocol definition that still needed to be worked out. Overall, the protocol that was written for our group entailed a more rigid, verbose protocol that was more transaction-based; that is, if one wanted to play a monopoly card or steal from a player, it was all done through one message. This was decided on as it was decidedly simpler than the alternative, in which state and expected moves had to then be maintained.

Contrastingly, the exact opposite approach was taken in the group protocol; requests have been broken up into their logical constituents, and are sent independently from one another as it, admittedly, better replicates the flow of a board game. For example, in "Settlers of Catan" when playing a year of plenty card, one would tend to play the card, and then subsequently decide on which resources they wished to receive, as opposed to doing it all at once. This methodology has been extended to every request type, and thus necessitated the maintenance of state and which moves are expected on both the client implementations and the Server. Due to the fundamental contrast in group approach and internal approach, the decision was made to simply migrate the existing code base entirely to the group protocol (once it became finalised enough). This in turn, involves a much more unified and intuitive code base, as much less translation between messages types is necessary.

AI logic

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

Proin non sem. Donec nec erat. Proin libero. Aliquam viverra arcu. Donec vitae purus. Donec felis mi, semper id, scelerisque porta, sollicitudin sed, turpis. Nulla in urna. Integer varius wisi non elit. Etiam nec sem. Mauris consequat, risus nec congue condimentum, ligula ligula suscipit urna, vitae porta odio erat quis sapien. Proin luctus leo id erat. Etiam massa metus, accumsan pellentesque, sagittis sit amet, venenatis nec, mauris.

Praesent urna eros, ornare nec, vulputate eget, cursus sed, justo. Phasellus nec lorem. Nullam ligula ligula, mollis sit amet, faucibus vel, eleifend ac, dui. Aliquam erat volutpat.

Fusce vehicula, tortor et gravida porttitor, metus nibh congue lorem, ut tempus purus mauris a pede. Integer tincidunt orci sit amet turpis. Aenean a metus. Aliquam vestibulum lobortis felis. Donec gravida. Sed sed urna. Mauris et orci. Integer ultrices feugiat ligula. Sed dignissim nibh a massa. Donec orci dui, tempor sed, tincidunt nonummy, viverra sit amet, turpis. Quisque lobortis. Proin venenatis tortor nec wisi. Vestibulum placerat. In hac habitasse platea dictumst. Aliquam porta mi quis risus. Donec sagittis luctus diam. Nam ipsum elit, imperdiet vitae, faucibus nec, fringilla eget, leo. Etiam quis dolor in sapien porttitor imperdiet.

Cras pretium. Nulla malesuada ipsum ut libero. Suspendisse gravida hendrerit tellus. Maecenas quis lacus. Morbi fringilla. Vestibulum odio turpis, tempor vitae, scelerisque a, dictum non, massa. Praesent erat felis, porta sit amet, condimentum sit amet, placerat et, turpis. Praesent placerat lacus a enim. Vestibulum non eros. Ut congue. Donec tristique varius tortor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nam dictum dictum urna.

Phasellus vestibulum orci vel mauris. Fusce quam leo, adipiscing ac, pulvinar eget, molestie sit amet, erat. Sed diam. Suspendisse eros leo, tempus eget, dapibus sit amet, tempus eu, arcu. Vestibulum wisi metus, dapibus vel, luctus sit amet, condimentum quis, leo. Suspendisse molestie. Duis in ante. Ut sodales sem sit amet mauris. Suspendisse ornare pretium orci. Fusce tristique enim eget mi. Vestibulum eros elit, gravida ac, pharetra sed, lobortis in, massa. Proin at dolor. Duis accumsan accumsan pede. Nullam blandit elit in magna lacinia hendrerit. Ut nonummy luctus eros. Fusce eget tortor.

Ut sit amet magna. Cras a ligula eu urna dignissim viverra. Nullam tempor leo porta ipsum. Praesent purus. Nullam consequat. Mauris dictum sagittis dui. Vestibulum sollicitudin consectetuer wisi. In sit amet diam. Nullam malesuada pharetra risus. Proin lacus arcu, eleifend sed, vehicula at, congue sit amet, sem. Sed sagittis pede a nisl. Sed tincidunt odio a pede. Sed dui. Nam eu enim. Aliquam sagittis lacus eget libero. Pellentesque diam sem, sagittis molestie, tristique et, fermentum ornare, nibh. Nulla et tellus non felis imperdiet mattis. Aliquam erat volutpat.

Evaluation and critical appraisal

Evaluating the work with respect to the original objectives. The section should also critically evaluate the work with respect to related work done by others. It should compare and contrast the project with similar work in the public domain, for example as written about in published papers, or as distributed in software available to the team.

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetuer mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

Nullam eleifend justo in nisl. In hac habitasse platea dictumst. Morbi nonummy. Aliquam ut felis. In velit leo, dictum vitae, posuere id, vulputate nec, ante. Maecenas vitae pede nec dui dignissim suscipit. Morbi magna. Vestibulum id purus eget velit laoreet laoreet. Praesent sed leo vel nibh convallis blandit. Ut rutrum. Donec nibh. Donec interdum. Fusce sed pede sit amet elit rhoncus ultrices. Nullam at enim vitae pede vehicula iaculis.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Aenean nonummy turpis id odio. Integer euismod imperdiet turpis. Ut nec leo nec diam imperdiet lacinia. Etiam eget lacus eget mi ultricies posuere. In placerat tristique tortor. Sed porta vestibulum metus. Nulla iaculis sollicitudin pede. Fusce luctus tellus in dolor. Curabitur auctor velit a sem. Morbi sapien. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Donec adipiscing urna vehicula nunc. Sed ornare leo in leo. In rhoncus leo ut dui. Aenean dolor quam, volutpat nec, fringilla id, consectetuer vel, pede.

Nulla malesuada risus ut urna. Aenean pretium velit sit amet metus. Duis iaculis. In hac habitasse platea dictumst. Nullam molestie turpis eget nisl. Duis a massa id pede dapibus ultricies. Sed eu leo. In at mauris sit amet tortor bibendum varius. Phasellus justo risus, posuere in, sagittis ac, varius vel, tortor. Quisque id enim. Phasellus consequat, libero pretium nonummy fringilla, tortor lacus vestibulum nunc, ut rhoncus ligula neque id justo. Nullam accumsan euismod nunc. Proin vitae ipsum ac metus dictum tempus. Nam ut wisi. Quisque tortor felis, interdum ac, sodales a, semper a, sem. Curabitur in velit sit amet dui tristique sodales. Vivamus mauris pede, lacinia eget, pellentesque quis, scelerisque eu, est. Aliquam risus. Quisque bibendum pede eu dolor.

Donec tempus neque vitae est. Aenean egestas odio sed risus ullamcorper ullamcorper. Sed in nulla a tortor tincidunt egestas. Nam sapien tortor, elementum sit amet, aliquam in, porttitor faucibus, enim. Nullam congue suscipit nibh. Quisque convallis. Praesent arcu nibh, vehicula eget, accumsan eu, tincidunt a, nibh. Suspendisse vulputate, tortor quis adipiscing viverra, lacus nibh dignissim tellus, eu suscipit risus ante fringilla diam. Quisque a libero vel pede imperdiet aliquet. Pellentesque nunc nibh, eleifend a, consequat consequat, hendrerit nec, diam. Sed urna. Maecenas laoreet eleifend neque. Vivamus purus odio, eleifend non, iaculis a, ultrices sit amet, urna. Mauris faucibus odio vitae risus. In nisl. Praesent purus. Integer iaculis, sem eu egestas lacinia, lacus pede scelerisque augue, in ullamcorper dolor eros ac lacus. Nunc in libero.

Fusce suscipit cursus sem. Vivamus risus mi, egestas ac, imperdiet varius, faucibus quis, leo. Aenean tincidunt. Donec suscipit. Cras id justo quis nibh scelerisque dignissim. Aliquam sagittis elementum dolor. Aenean consectetuer justo in pede. Curabitur ullamcorper ligula nec orci. Aliquam purus turpis, aliquam id, ornare vitae, porttitor non, wisi. Maecenas luctus porta lorem. Donec vitae ligula eu ante pretium varius. Proin tortor metus, convallis et, hendrerit non, scelerisque in, urna. Cras quis libero eu ligula bibendum tempor. Vivamus tellus quam, malesuada eu, tempus sed, tempor sed, velit. Donec lacinia auctor libero.

Praesent sed neque id pede mollis rutrum. Vestibulum iaculis risus. Pellentesque lacus. Ut quis nunc sed odio malesuada egestas. Duis a magna sit amet ligula tristique pretium. Ut pharetra. Vestibulum imperdiet magna nec wisi. Mauris convallis. Sed accumsan sollicitudin massa. Sed id enim. Nunc pede enim, lacinia ut, pulvinar quis, suscipit semper, elit. Cras accumsan erat vitae enim. Cras sollicitudin. Vestibulum rutrum blandit massa.

Sed gravida lectus ut purus. Morbi laoreet magna. Pellentesque eu wisi. Proin turpis. Integer sollicitudin augue nec dui. Fusce lectus. Vivamus faucibus nulla nec lacus. Integer diam. Pellentesque sodales, enim feugiat cursus volutpat, sem mauris dignissim mauris, quis consequat sem est fermentum ligula. Nullam justo lectus, condimentum sit amet, posuere a, fringilla mollis, felis. Morbi nulla nibh, pellentesque at, nonummy eu, sollicitudin nec, ipsum. Cras neque. Nunc augue. Nullam vitae quam id quam pulvinar blandit. Nunc sit amet orci. Aliquam erat elit, pharetra nec, aliquet a, gravida in, mi. Quisque urna enim, viverra quis, suscipit quis, tincidunt ut, sapien. Cras placerat consequat sem. Curabitur ac diam. Curabitur diam tortor, mollis et, viverra ac, tempus vel, metus.

Curabitur ac lorem. Vivamus non justo in dui mattis posuere. Etiam accumsan ligula id pede. Maecenas tincidunt diam nec velit. Praesent convallis sapien ac est. Aliquam ullamcorper euismod nulla. Integer mollis enim vel tortor. Nulla sodales placerat nunc. Sed tempus rutrum wisi. Duis accumsan gravida purus. Nunc nunc. Etiam facilisis dui eu sem. Vestibulum semper. Praesent eu eros. Vestibulum tellus nisl, dapibus id, vestibulum sit amet, placerat ac, mauris. Maecenas et elit ut erat placerat dictum. Nam feugiat, turpis et sodales volutpat, wisi quam rhoncus neque, vitae aliquam ipsum sapien vel enim. Maecenas suscipit cursus mi.

Quisque consectetuer. In suscipit mauris a dolor pellentesque consectetuer. Mauris convallis neque non erat. In lacinia. Pellentesque leo eros, sagittis quis, fermentum quis, tincidunt ut, sapien. Maecenas sem. Curabitur eros odio, interdum eu, feugiat eu, porta ac, nisl. Curabitur nunc. Etiam fermentum convallis velit. Pellentesque laoreet lacus. Quisque sed elit. Nam quis tellus. Aliquam tellus arcu, adipiscing non, tincidunt eleifend, adipiscing quis, augue. Vivamus elementum placerat enim. Suspendisse ut tortor. Integer faucibus adipiscing felis. Aenean consectetuer mattis lectus. Morbi malesuada faucibus dolor. Nam lacus. Etiam arcu libero, malesuada vitae, aliquam vitae, blandit tristique, nisl.

Maecenas accumsan dapibus sapien. Duis pretium iaculis arcu. Curabitur ut lacus. Aliquam vulputate. Suspendisse ut purus sed sem tempor rhoncus. Ut quam dui, fringilla at, dictum eget, ultricies quis, quam. Etiam sem est, pharetra non, vulputate in, pretium at, ipsum. Nunc semper sagittis orci. Sed scelerisque suscipit diam. Ut volutpat, dolor at ullamcorper tristique, eros purus mollis quam, sit amet ornare ante nunc et enim.

Phasellus fringilla, metus id feugiat consectetuer, lacus wisi ultrices tellus, quis lobortis nibh lorem quis tortor. Donec egestas ornare nulla. Mauris mi tellus, porta faucibus, dictum vel, nonummy in, est. Aliquam erat volutpat. In tellus magna, porttitor lacinia, molestie vitae, pellentesque eu, justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed orci nibh, scelerisque sit amet, suscipit sed, placerat vel, diam. Vestibulum nonummy vulputate orci. Donec et velit ac arcu interdum semper. Morbi pede orci, cursus ac, elementum non, vehicula ut, lacus. Cras volutpat. Nam vel wisi quis libero venenatis placerat. Aenean sed odio. Quisque posuere purus ac orci. Vivamus odio. Vivamus varius, nulla sit amet semper viverra, odio mauris consequat lacus, at vestibulum neque arcu eu tortor. Donec iaculis tincidunt tellus. Aliquam erat volutpat. Curabitur magna lorem, dignissim volutpat, viverra et, adipiscing nec, dolor. Praesent lacus mauris, dapibus vitae, sollicitudin sit amet, nonummy eget, ligula.

Cras egestas ipsum a nisl. Vivamus varius dolor ut dolor. Fusce vel enim. Pellentesque accumsan ligula et eros. Cras id lacus non tortor facilisis facilisis. Etiam nisl elit, cursus sed, fringilla in, congue nec, urna. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer at turpis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis fringilla, ligula sed porta fringilla, ligula wisi commodo felis, ut adipiscing felis dui in enim. Suspendisse malesuada ultrices ante. Pellentesque scelerisque augue sit amet urna. Nulla volutpat aliquet tortor. Cras aliquam, tellus at aliquet pellentesque, justo sapien commodo leo, id rhoncus sapien quam at erat. Nulla commodo, wisi eget sollicitudin pretium, orci orci aliquam orci, ut cursus turpis justo et lacus. Nulla vel tortor. Quisque erat elit, viverra sit amet, sagittis eget, porta sit amet, lacus.

Conclusions

Summarising the project, emphasising key achievements and significant drawbacks, and discussing possible future directions for the work.

In hac habitasse platea dictumst. Proin at est. Curabitur tempus vulputate elit. Pellentesque sem. Praesent eu sapien. Duis elit magna, aliquet at, tempus sed, vehicula non, enim. Morbi viverra arcu nec purus. Vivamus fringilla, enim et commodo malesuada, tortor metus elementum ligula, nec aliquet est sapien ut lectus. Aliquam mi. Ut nec elit. Fusce euismod luctus tellus. Curabitur scelerisque. Nullam purus. Nam ultricies accumsan magna. Morbi pulvinar lorem sit amet ipsum. Donec ut justo vitae nibh mollis congue. Fusce quis diam. Praesent tempus eros ut quam.

Donec in nisl. Fusce vitae est. Vivamus ante ante, mattis laoreet, posuere eget, congue vel, nunc. Fusce sem. Nam vel orci eu eros viverra luctus. Pellentesque sit amet augue. Nunc sit amet ipsum et lacus varius nonummy. Integer rutrum sem eget wisi. Aenean eu sapien. Quisque ornare dignissim mi. Duis a urna vel risus pharetra imperdiet. Suspendisse potenti.

Morbi justo. Aenean nec dolor. In hac habitasse platea dictumst. Proin nonummy porttitor velit. Sed sit amet leo nec metus rhoncus varius. Cras ante. Vestibulum commodo sem tincidunt massa. Nam justo. Aenean luctus, felis et condimentum lacinia, lectus enim pulvinar purus, non porta velit nisl sed eros. Suspendisse consequat. Mauris a dui et tortor mattis pretium. Sed nulla metus, volutpat id, aliquam eget, ullamcorper ut, ipsum. Morbi eu nunc. Praesent pretium. Duis aliquam pulvinar ligula. Ut blandit egestas justo. Quisque posuere metus viverra pede.

Vivamus sodales elementum neque. Vivamus dignissim accumsan neque. Sed at enim. Vestibulum nonummy interdum purus. Mauris ornare velit id nibh pretium ultricies. Fusce tempor pellentesque odio. Vivamus augue purus, laoreet in, scelerisque vel, commodo id, wisi. Duis enim. Nulla interdum, nunc eu semper eleifend, enim dolor pretium elit, ut commodo ligula nisl a est. Vivamus ante. Nulla leo massa, posuere nec, volutpat vitae, rhoncus eu, magna.

Quisque facilisis auctor sapien. Pellentesque gravida hendrerit lectus. Mauris rutrum sodales sapien. Fusce hendrerit sem vel lorem. Integer pellentesque massa vel augue. Integer elit tortor, feugiat quis, sagittis et, ornare non, lacus. Vestibulum posuere pellentesque eros. Quisque venenatis ipsum dictum nulla. Aliquam quis quam non metus eleifend interdum. Nam eget sapien ac mauris malesuada adipiscing. Etiam eleifend neque sed quam. Nulla facilisi. Proin a ligula. Sed id dui eu nibh egestas tincidunt. Suspendisse arcu.

Maecenas dui. Aliquam volutpat auctor lorem. Cras placerat est vitae lectus. Curabitur massa lectus, rutrum euismod, dignissim ut, dapibus a, odio. Ut eros erat, vulputate ut, interdum non, porta eu, erat. Cras fermentum, felis in porta congue, velit leo facilisis odio, vitae consectetuer lorem quam vitae orci. Sed ultrices, pede eu placerat auctor, ante ligula rutrum tellus, vel posuere nibh lacus nec nibh. Maecenas laoreet dolor at enim. Donec molestie dolor nec metus. Vestibulum libero. Sed quis erat. Sed tristique. Duis pede leo, fermentum quis, consectetuer eget, vulputate sit amet, erat.

Donec vitae velit. Suspendisse porta fermentum mauris. Ut vel nunc non mauris pharetra varius. Duis consequat libero quis urna. Maecenas at ante. Vivamus varius, wisi sed egestas tristique, odio wisi luctus nulla, lobortis dictum dolor ligula in lacus. Vivamus aliquam, urna sed interdum porttitor, metus orci interdum odio, sit amet euismod lectus felis et leo. Praesent ac wisi. Nam suscipit vestibulum sem. Praesent eu ipsum vitae pede cursus venenatis. Duis sed odio. Vestibulum eleifend. Nulla ut massa. Proin rutrum mattis sapien. Curabitur dictum gravida ante.

Phasellus placerat vulputate quam. Maecenas at tellus. Pellentesque neque diam, dignissim ac, venenatis vitae, consequat ut, lacus. Nam nibh. Vestibulum fringilla arcu mollis arcu. Sed et turpis. Donec sem tellus, volutpat et, varius eu, commodo sed, lectus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Quisque enim arcu, suscipit nec, tempus at, imperdiet vel, metus. Morbi volutpat purus at erat. Donec dignissim, sem id semper tempus, nibh massa eleifend turpis, sed pellentesque wisi purus sed libero. Nullam lobortis tortor vel risus. Pellentesque consequat nulla eu tellus. Donec velit. Aliquam fermentum, wisi ac rhoncus iaculis, tellus nunc malesuada orci, quis volutpat dui magna id mi. Nunc vel ante. Duis vitae lacus. Cras nec ipsum.

Morbi nunc. Aliquam consectetuer varius nulla. Phasellus eros. Cras dapibus porttitor risus. Maecenas ultrices mi sed diam. Praesent gravida velit at elit vehicula porttitor. Phasellus nisl mi, sagittis ac, pulvinar id, gravida sit amet, erat. Vestibulum est. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Curabitur id sem elementum leo rutrum hendrerit. Ut at mi. Donec tincidunt faucibus massa. Sed turpis quam, sollicitudin a, hendrerit eget, pretium ut, nisl. Duis hendrerit ligula. Nunc pulvinar congue urna.

Nunc velit. Nullam elit sapien, eleifend eu, commodo nec, semper sit amet, elit. Nulla lectus risus, condimentum ut, laoreet eget, viverra nec, odio. Proin lobortis. Curabitur dictum arcu vel wisi. Cras id nulla venenatis tortor congue ultrices. Pellentesque eget pede. Sed eleifend sagittis elit. Nam sed tellus sit amet lectus ullamcorper tristique. Mauris enim sem, tristique eu, accumsan at, scelerisque vulputate, neque. Quisque lacus. Donec et ipsum sit amet elit nonummy aliquet. Sed viverra nisl at sem. Nam diam. Mauris ut dolor. Curabitur ornare tortor cursus velit.

Appendices

Project objectives, specification and initial plan

As submitted during the year.

Interim report

As submitted during the year.

Testing summary

Describing the steps taken to debug, test, verify or otherwise confirm the correctness of the various modules and their combination. To test this project, an entire unit test suite was implemented which tests the core backend, functionality and game logic, as well as if the server and client are successfully able to parse a message from the intergroup protocol and deal with it accordingly. There are countless tests in the backend which ensure that the correct exceptions get thrown when they ought to (for edge cases), as well as that “normal” use cases are covered as well. This ensures that all the game logic works as it should on the server side, and that the client is able to update its game state accordingly.

Aside from pure functionality testing, countless testing was done with AIs, clients and servers, to ensure that everything could communicate appropriately given all circumstances, expected and unexpected. AIs were tested using the main method in LocalServer.java, which simply starts up an off-line server with four AIs running directly off of it. This was used to ensure that AIs could successfully play through the initial phase with no issues. This also allowed tweaks and polish to be done to the move processing units of both the client and server, as the way in which expected moves were set up needed to be perfect to avoid errors or “hanging” AIs; this occurred if the server expected a certain move while the AI in question didn’t realise. This meant the server would prevent the entire game from proceeding until this given move was received, which it obviously never would be. With all these bugs sorted and tested thoroughly for validity and reproducibility, it could be said with certainty that the application as a whole has been throughly tested.

User manual

Instructions on installing, executing and using the system where appropriate.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Nam quis enim. Quisque ornare dui a tortor. Fusce consequat lacus pellentesque metus. Duis euismod. Duis non quam. Maecenas vitae dolor in ipsum auctor vehicula. Vivamus nec nibh eget wisi varius pulvinar. Cras a lacus. Etiam et massa. Donec in nisl sit amet dui imperdiet vestibulum. Duis porttitor nibh id eros.

Mauris consectetuer, wisi eu lobortis scelerisque, urna nibh feugiat quam, id congue eros justo eget orci. Ut tellus. Maecenas mattis sapien sed eros. Aliquam quis lectus. Donec nec massa ac turpis semper cursus. Etiam consectetuer ante vel odio. Aliquam tincidunt felis non dolor. Cras id augue ut nisl pretium placerat. Phasellus sapien sapien, pharetra sed, aliquam nec, suscipit a, nibh. Suspendisse risus. Nulla ut mi eget tellus sollicitudin euismod. Vestibulum malesuada malesuada dui. Ut at est ac dui aliquam sagittis. Aliquam erat volutpat.

Curabitur ullamcorper est in mauris. Praesent ac massa. Quisque enim odio, lobortis nec, mattis ut, luctus et, mauris. Mauris eu risus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eu ligula. Nulla vehicula leo tincidunt erat. Maecenas et nunc. Sed ut sapien. Vestibulum in est. Vestibulum rhoncus.

Donec metus metus, condimentum eu, accumsan nec, vulputate non, purus. Vestibulum ullamcorper vehicula sapien. Mauris risus odio, hendrerit ac, congue ac, ullamcorper at, odio. Aenean leo justo, commodo vitae, placerat blandit, malesuada vel, sem. Donec sit amet ante eget mauris adipiscing sollicitudin. Curabitur posuere sem et leo. Nulla ultricies mauris. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Fusce sollicitudin augue vel tellus. Vivamus mauris eros, pharetra vel, lacinia pretium, egestas a, nibh. Morbi a ligula.

Donec vitae turpis. Suspendisse porttitor. Mauris aliquam purus vitae tellus. Morbi metus diam, tempus ac, cursus ut, ultricies quis, nulla. Praesent nec justo. In lobortis. Donec nec lectus a neque laoreet rhoncus. Quisque in risus nec wisi lacinia ullamcorper. In placerat. Proin facilisis sollicitudin libero. Integer eget neque et pede placerat aliquet. Aliquam purus nulla, pulvinar ut, facilisis quis, sodales sed, magna. Curabitur nulla lectus, rutrum id, bibendum ut, sagittis eget, diam. Sed porta dolor eget est. Integer hendrerit orci. In hac habitasse platea dictumst.