Module Guide for SE 4G06
An AI-based Approach to Designing Board Games

Team #6, Board Gamers
Ilao Michael, ilaom
Bedi Hargun, bedih
Dang Jeffrey, dangj12
Ada Jonah, karaatan
Mai Tianzheng, mait6

January 18, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AI	Artifical Intellegence
GE	Game Engine
DV	Data Visualizer
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
SE 4G06	Explanation of program name
UC	Unlikely Change

2.2 Guideline

This Design Document, Capstone Intelligence was used as a template and guideline for this document.

Contents

List of Tables

List of Figures

3 Purpose

3.1 System Purpose

The purpose of the system is to simulate the board game An Age Contrived from Bellow Intent's using an AI-based game simulation engine. This system will help identify pitfall's in game mechanics and imbalances in the game's scoring system. The system will also be used to view decisions, scoring and other game state information through various plots, graphs and charts. The system will be used by Game Designers of An Age Contrived to help balance their game as well as AI Research Professors to further understand and develop AI-based game engines.

3.2 Document Purpose

The purpose of this document is to decompose the system into different modules, to show the module and system architecture and provide justifications for each module. This will serve as a guideline for development of the system.

4 Scope

The system is to be used by game designers when creating and balancing their board games and AI researchers

4.1 Assumptions

A1: The game to be analyzed will have a clear reward function that AI Agent can optimize its learning towards.

A2: The game to be analyzed will have a clear win condition where the game ends.
Rationale: It shouldn't be an open-ended, infinite game where the goal is to just increase your score but rather there should be a specific condition(s) that players are playing towards to win and end the game. Otherwise, AI agents won't be able to find the optimal actions to win the game but rather keep outputting sub-optimal decisions which will have little to no value for game design improvements.

A3: The game to be analyzed will have concrete rules that a computer can represent and enforce.
Rationale: If the game cannot be simulated digitally, then AI Agent cannot be trained on the game and the system wouldn't be able to generate any meaningful data to aid the game design process.

5 Project Overview

5.1 Normal Operation

This project will be used by board game designers to rapid prototype their development of the game rules and game metrics. Board game designers will be able to evaluate the balance of their game in terms if there exists an always winning sequence of moves that are not intended to be that strong to achieve the win condition in their game. Simulating many games through AI players will save a large amount of time in testing by analyzing the moves and winning sequences of the large data set of games they will generate.

5.2 Behaviour Overview

The system is designed on a command pattern architecture. When the simulation is run, AI Agents are able to choose a selection of moves in. After a game move is chosen, the game engine will accept and change its state based on the move given. This will feedback on where the game engine will give the game state back to the AI Agents, which they can choose another action after receiving the game state. This will repeat until the end of the game is reached.

5.3 Module Hierarchy

Module Type	Module Name	Module Description
AI	AI Agent Module	Trains AI Agents on the game and generates a policy
AI	Game Environment Module	Receives input from AI Agents to take action on the game
GE	Actions (Commands) Module	Describes the possible game moves that the AI Agents are able to take
GE	Transmuter Module	Controls all actions related to transmuters such as, getting all tiles, conveying tiles, filling tiles into the transmuter, and printing transmuter to console.
GE	Monument Module	Responsible for all action related to monument control such as, checking for monument completion, changing the top monument wall, and returning the completed monuments.
GE	Energy Module	Initializes energy tile types and coordinates every tile to a player
GE	Player Module	Initializes a player that has a specific character, its own transmuters, energies, and player name.
GE	Game Engine Module	Continues the game loop for the game and checks if the game-over condition has been fulfilled or not.
DV	JSON Module	Responsible for recording each AI Agents moves and observation space and putting them into a JSON file.
DV	Pie Chart Module	Produce a Pie Chart Data model to show percentages of different objects.
DV	Bar Chart Module	Responsible to show a distribution of data points and perform a comparison of different metric values.
DV	Line Chart Module	Responsible to connect a series of data points and perform a comparison.
DV	JSONDataParser Module	Parses JSON files with AI Agents' move history

Table 1: Module Hierarchy

5.4 Undesired Event Handling

When the system experiences an undesired event, the system will immediately end any simulation and continue processing a new simulation. The data recorded from that undesired event will be stored in the JSON file and can be analyzed for future reference to understand what went wrong in that simulation.

5.5 System Context Diagram

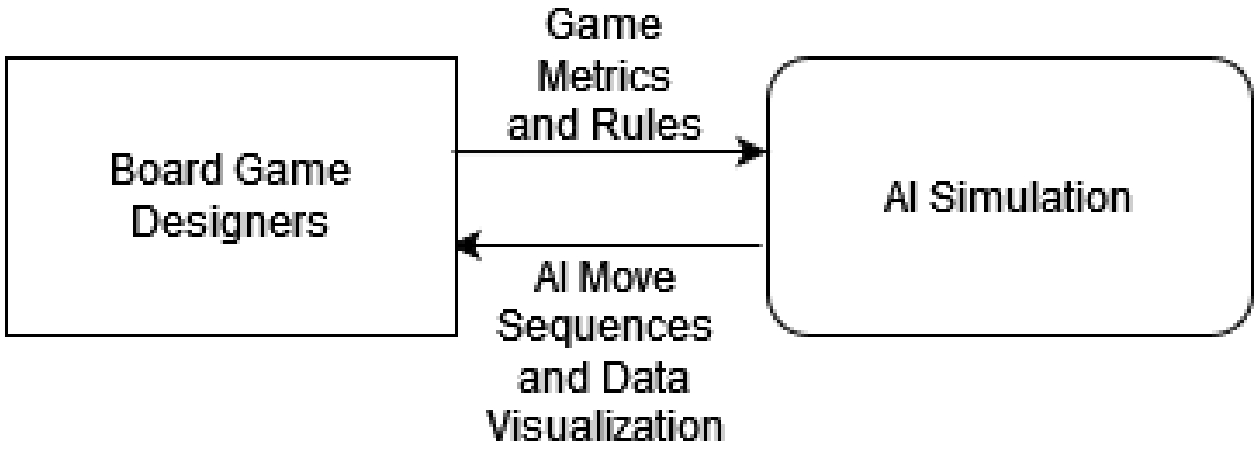

Figure 1: System Context Diagram

5.6 Component Diagram

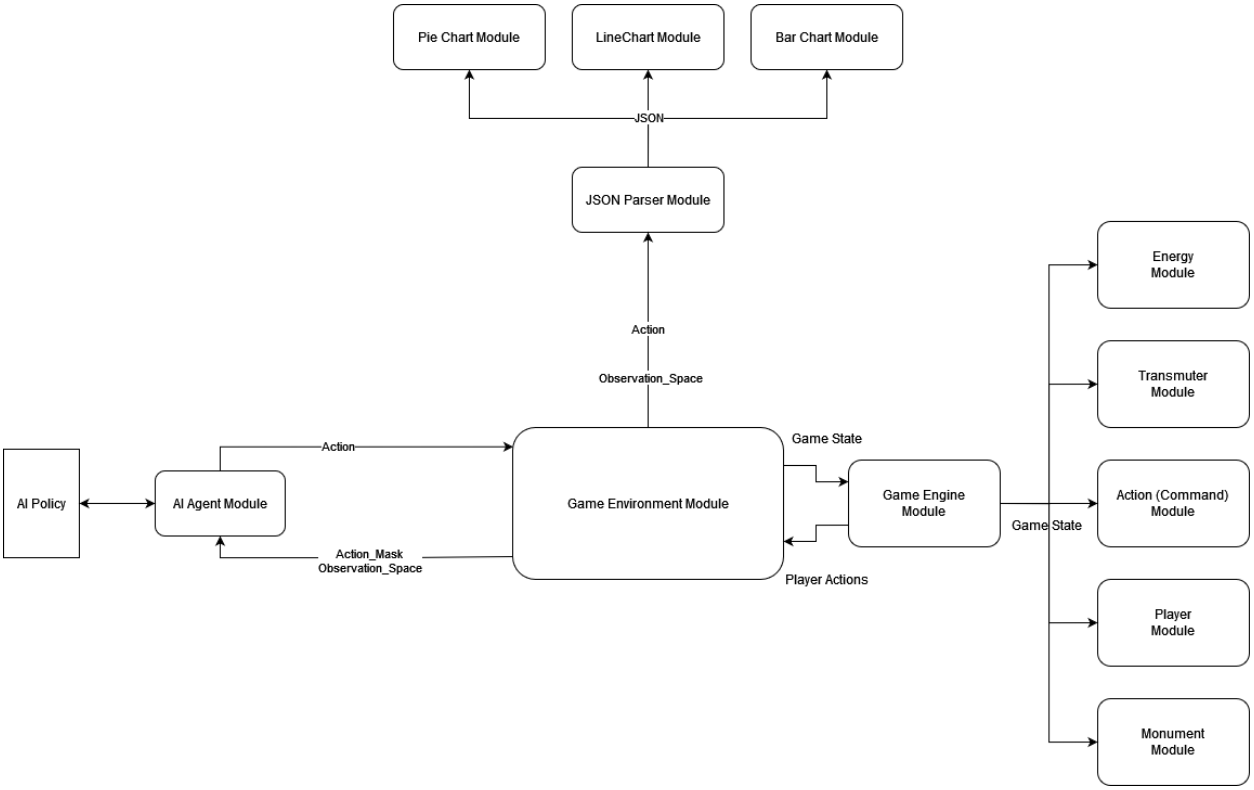

Figure 2: Component Diagram

6 System Variables and Notation

6.1 Monitored Variables

Monitored variables will be any variables that are observed by the system that change due to the controlled variables.

6.2 Controlled Variables

Controlled variables will be any variables that are controlled by the system to change the monitored variables. Refer to the output in each modules for the controlled variables.

6.3 Constants

There are no constants.

6.4 Notation

Type	Description
$\mathbb{N}$	Real Numbers ≥ 0
String	sequence of characters
List$\langle Type \rangle$	An array or list of the specified type
Object	An instansiation of a module or abstract data type.

7 Module Traceability

Module Name	**Requirements**
AI Agent Module	FR1 FR2 FR4 FR6 NFR1 NFR3 NFR4 NFR5
Game Environment Module	FR3 FR5 NFR1 NFR3 NFR5
Action (Command) Module	FR4 FR7 FR11 NFR3 NFR6
Transmuter Module	FR10
Monument Module	FR10
Energy Module	FR10
Player Module	FR10
Game Engine Module	FR7 FR8 FR9 FR11 NFR1 NFR3 NFR4 NFR5 NFR6
JSON Module	FR3 FR13 NFR3
Pie Chart Module	FR12 NFR2 NFR7
Bar Chart Module	FR12 NFR2 NFR7
Line Chart Module	FR12 NFR2 NFR7
JSONDataParser Module	FR13 FR14 FR15

Table 2: Module Traceability

8 Abstract Data Types

8.1 TransmuterTile

8.1.1 Inputs

Name	Value Type	Description
Top_Size	$\mathbb{N}$	This value corresponds to the number of Energy sections in top half of the tile.
Bottom_Size	$\mathbb{N}$	This value corresponds to the number of Energy sections in bottom half of the tile.

8.1.2 Outputs

Name	Value Type	Description
TransmuterTile	*Object* TransmuterTile	Initialized instance of a TransmuterTile type with the specified input parameters

8.1.3 Description of Behaviour

This data type is a representation of the Transmuter Tile on the player's transmuter board. It contains a method to fill the tile with an energy at a specified position on the tile and a method to empty all the energy placed on the tile. The input is used to determine the size of the top and bottom half of the tile where the energy can be placed.

8.1.4 Exception Handling

Input validation will occur at the beginning of the module initialization to handle exceptions to see if any violations occur (e.g. out of bounds error). Error handling will also be done in the methods to fill a position in the tile to validate the position index and Energy type.

8.1.5 Initialization

Multiple TransmuterTiles will be initialized in the Transmuter module.

8.2 MonumentWall

8.2.1 Inputs

Name	Value Type	Description
Acceptable_Energies	List⟨*Energy*⟩	As each wall can accept different energy types, it is necessary to input the acceptable energy types for a wall
Benefit_Token	String	Once a wall is completed, the wall owner will receive this benefit token allowing for special abilities.

8.2.2 Outputs

Name	Value Type	Description
MonumentWall	*Object* MonumentWall	Initialized instance of a MonumentWall type with the specified input parameters

8.2.3 Description of Behaviour

This data type is a representation of the Monument Wall tile on the game board. It has methods to fill an energy section and attributes to store the acceptable energy types, the current state of the wall (completed/not completed) and the benefit token. Once the energies are filled, an owner (player) will be assigned to the tile which will be useful for calculating the score at the end of the game.

8.2.4 Exception Handling

Input validation will occur at the beginning of the module initialization to handle exceptions to see if any violations occur (e.g. out of bounds error, type error). Error handling will be done in methods to fill a wall section to validate the input Energy type and position index as well as if the wall hasn't already been filled.

8.2.5 Initialization

Multiple MonumentWalls will be initialized in the Game Environment module.

9 Module Guide

9.1 AI Agent Module

9.1.1 Inputs

Input Name	Input Type	Description
Epoch	$\mathbb{N}$	Number of epochs to run.
Batch_Size	$\mathbb{N}$	Size of batch per epoch
Resume_Path	String	Path to policy for the main agent. (This is the agent that the policy generated will be modeled from)
Opponent_Path	List$\langle String \rangle$	Path to the policies for each of the opponents
Gamma	Float	Learning rate (smaller gamma, the AI will try to win earlier)
Agent_Id	$\mathbb{N}$	The ID to set for the main agent

9.1.2 Outputs

Output Name	Output Type	Description
Policy	$\mathbb{N} * \mathbb{N}$	Internal filetype of AI library that stores the policy of a training simulation. $\mathbb{N}$ will vary depending on Observation_Space and Action_Space

9.1.3 Description of Behaviour

This module will be the initial entry point into the system. After entering the inputs the module will initialize the AI Agents and their respective policies and the **GameEnvironment**, that the AI Agents will interact with. The module will map the policies using the Resume_Path for the Agent with the Id from Agent_Id, and map the rest with the Opponent_Path policies. Once the AI Agents and environment is ready the training will run for Epoch simulations, and during each simulation the game will be run equal to the Batch_Size. In total the number of individual games run will be Epoch∗Batch_Size. Once the training is complete the module will generate a policy based off these simulations and run an additional time rendering the game simulation for the user to view the AI Agent's make decisions.

9.1.4 Exception Handling

Input validation will occur at the beginning of the module initialization to handle exceptions that will terminate runtime if any violations occur (type mismatch or not within bounds). Validation will also check to ensure that the paths provided point to valid files.

9.1.5 Initialization

Upon starting the system the AI Agent is the first to be initialized creating the AI Agents with their corresponding policies and the game environment they will interact with.

9.2 Game Environment Module

9.2.1 Inputs

Input Name	Input Type	Description
Agents	List$\langle String \rangle$	List of agent names
Action	$\mathbb{N}$	An array index that corresponds to the action that the current AI Agent will play.

9.2.2 Outputs

Output Name	Output Type	Description
Rewards	List$\langle Float \rangle$	The associated award for each agent during a single game simulation
Observation_Space	$\mathbb{N} * \mathbb{N}$	Observable state of the game that is updated upon each action taken. This is a matrix of $\mathbb{N} * \mathbb{N}$
Action_Space	$\mathbb{N}$	An array of all possible actions that can be taken of size $\mathbb{N}$
Action_Mask	$\mathbb{N}$	An array of all legal actions of size $\mathbb{N}$, an agent can take, given a specific Observation_Space

9.2.3 Description of Behaviour

This module is the bridge between the AI Agents and the pure Game Engine. It will facilitate the input of an action taken by the AI and ensuring it is a legal move, while providing an observable state for policy learning. The module will hold information on each AI Agent's rewards to better choose which actions are better to take at certain game states.

9.2.4 Exception Handling

Exception handling is done by the AI Game Environment library, where if an exception occurs like an illegal action is taken the simulation will terminate and a negative reward will be assigned to the AI. If the AI chooses an action that is out of bound (Action is higher than the length of the Action_Space), the module will throw an error. If the Action_Space

is not in the same order during simulations an error will be thrown as these are needed for successful simulations.

9.2.5 Initialization

This module is initialized after the AI Agents are initialized on the AI_Agent Module and only initialized once. This module will then initialize the game engine along with the Observable_Space, Action_Space, and Rewards. Upon finishing a game simulation the module is reset and Rewards are then passed back to the AI_Agent module.

9.3 Game Engine Module

9.3.1 Inputs

Game Engine has no explicit inputs to initialize it but rather it is the starting point of the all business logic. It is initialized to initialize the rest of the system. However, it has many dependencies on other modules to initialize them and to enforce the core business logic of the system. It uses the following modules explicitly:

- Player Module
- Monument Module
- Transmuter Module
- EnergyTile Module
- ActionInitiater Module

9.3.2 Outputs

Output Name	Output Type	Description
Engine	*Object* Game Engine	Initialized instance of a Game Engine module. Drives the business logic and the entry point to the game.

9.3.3 Description of Behaviour

This module is where the entire business logic is written and it is the entry point to the game. It uses many of the core modules of the system to initialize a new game engine and start a new simulation. Than interfacing with the Game Environment module, it provides the current state of the game and possible actions to the AI Agents and processes their response to drive the game. It also dictates the turn order of the players and keeps track of game termination conditions.

9.3.4 Exception Handling

There are many exception handling logic embedded inside the Game Engine module to ensure that the Player Module, Monument Module, Transmuter Module, EnergyTile Module and other aspects of the game is initialized correctly and throughout the game they are following the correct business logic. However, after initialization, if there is an error occurred, since the current simulation is not mission critical and we can gather the same data the in the next simulation, if the error handling cannot handle it and requires a human intervention, since the AI Agents won't be able to fix the issue, the data will be discarded and a new simulation will be started. This ensure the data collected is always reliable instead of trying to handle some complex errors based on the 40+ pages of game rules (business logic).

9.3.5 Initialization

This module is initialized when the user of the system first enters the command to run the system to the terminal. There is a __main__.py file that defines the entry point of the system which directly calls the Game Engine module to initialize it.

9.4 Player Module

9.4.1 Inputs

Name	Value Type	Description
Character	String	Name of Character
Agent_Id	$\mathbb{N}$	The ID to set for the main agent
Energies	List$\langle EnergyTile \rangle$	The list of EnergyTiles that a Player starts with

9.4.2 Outputs

Name	Value Type	Description
Player	*Object* Player	Initialized instance of a Player module with the specified input parameters

9.4.3 Description of Behaviour

This module contains attributes and methods that will be used to represent a Player. At initialization, an instance of Transmuter will also be initialized. It will contain methods that will support updating the Transmuter and its TransmuterTiles. This module will also store the special abilites that a player will have based on the character name provided. It will also allow the Player to add and remove EnergyTiles in the Energies attribute.

9.4.4 Exception Handling

Exception handling will be done during initialization of the Player to validate the character name, list of EnergyTiles, and Agent_Id. Additionally, any mutator methods will also have error handling to validate the input parameters and will throw the appropriate error.

9.4.5 Initialization

A Player will be initialized in the AI Agent module with the required input parameters.

9.5 Transmuter Module

9.5.1 Inputs

No inputs for this module.

9.5.2 Outputs

Name	Value Type	Description
Transmuter	*Object* Transmuter	Initialized instance of a Transmuter module with the specified input parameters
Active_Tiles	List$\langle TransmuterTile \rangle$	List of TransmuterTiles that are currently on the Transmuter
Reserved_Tiles	List$\langle TransmuterTile \rangle$	List of TransmuterTiles that are currently on reserve.
Action_Tokens	List$\langle String \rangle$	List of tokens that corresponds with a position on the Active_Tiles. Tokens represent a special ability.

9.5.3 Description of Behaviour

This module contains attributes and methods to represent a player's Transmuter board. As all players have the same starting TransumterTiles, there is no input for this module. The Transmuter behaves like a queue as it contains methods to convey the Active_Tiles which removes the last active TransmuterTile and adds a TransmuterTile from the Reserved_Tiles list to the Active_Tiles. It will also contain methods to add, retrieve and update the Action_Token at a particular index on the Transmuter.

9.5.4 Exception Handling

Exception handling will be done in methods that require inputs that will alter the state variables. Errors will be thrown if the input is in an unexpected format or if the input is invalid (e.g. list index out of bounds). Getter methods will not require any exception handling.

9.5.5 Initialization

Upon initializing a Player, a Transmuter will be initialized as well.

9.6 Monument Module

9.6.1 Inputs

Name	Value Type	Description
Monument_Walls	List$\langle MonumentWall \rangle$	List of MonumentWalls that are on on the Monument
Name	String	Name of the Monument.

9.6.2 Outputs

Name	Value Type	Description
Monument	*Object* Monument	Initialized instance of a Monument module with the specified input parameters
Top_Wall_Index	$\mathbb{N}$	Index of the top MonumentWall on the Transmuter
Completed_Tiles	List$\langle MonumentWalls \rangle$	List of TransmuterTiles that are currently on reserve.

9.6.3 Description of Behaviour

This module contains methods and attributes that represent a Monument. As each Monument can have different variations of MonumentWalls, a list of MonumentWalls is given as input to this module. It has methods to change, update and retrieve the current top MonumentWall, in accordance with the game rules. As the MonumentWalls get completed, they get added to the Completed_Tiles output list.

9.6.4 Exception Handling

Exception Handling will be done during the initialization of the module to validate the input. It is necessary that the Monument_Walls is of List$\langle MonumentWall \rangle$ type. If the type is incorrect or if the size of the list is incorrect, an error will be thrown.

9.6.5 Initialization

Upon starting the Game Engine, multiple Monuments will be initialized, in accordance with the game rules.

9.7 EnergyTile Module

9.7.1 Inputs

Name	Value Type	Description
Energy_Type	Enum	Type of the energy which is enumerated and as follows: [Constructive: 1, Invertible: 2, Generative: 3, Primal: 4]
Owner	Player	Player which owns the energy. Each energy tile can only have one owner.

9.7.2 Outputs

Name	Value Type	Description
EnergyTile	*Object* EnergyTile	Initialized instance of a EnergyTile module with one of the accepted energy types

9.7.3 Description of Behaviour

This module contains methods and attributes that represent Energy in the game. There are 4 different types of energies [Constructive: 1, Invertible: 2, Generative: 3, Primal: 4] and they are enumerated to be used throughout the system. Each energy type has different uses in the game but Primal energy type is a special one which can be used in place of other three energy types. Each energy also needs to belong to a player which is being monitored via owner variable. Methods are only getters which returns back the energy type and the owner of the energy.

9.7.4 Exception Handling

If the given energy type is not in the enumerated list of correct energy types, exception handling will kick in and print an error message as well as setting the energy type as invalid which would prevent it to be used during the game.

9.7.5 Initialization

When initialized, EnergyTile module will return an instance of EnergyTile to be used by players to make decisions and take actions based on the game rules.

9.8 Command Module

9.8.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.8.2 Outputs

Name	Value Type	Description
Command	*Object* Command	Initialized instance of a command object.

9.8.3 Description of Behaviour

This module is a parent class to all action classes. It is named Command because it is the parent class that helps the system to implement the Command Design Pattern.

9.8.4 Exception Handling

N/A

9.8.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.9 ActionTurn Module inherits Command Module

9.9.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.9.2 Outputs

Name	Value Type	Description
ActionTurn	*Object* Command	Initialized instance of a command object. Allows users to select the turn type as action turn.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.9.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to select the turn to be action turn instead of the convey turn.

9.9.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.9.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.10 ConveyTurn Module inherits Command Module

9.10.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.10.2 Outputs

Name	Value Type	Description
ConveyTurn	*Object* Command	Initialized instance of a command object. Allows users to select the turn type as convey turn.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.10.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to select the turn to be convey turn instead of the action turn.

9.10.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.10.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.11 EndTurn Module inherits Command Module

9.11.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.11.2 Outputs

Name	Value Type	Description
EndTurn	*Object* Command	Initialized instance of a command object. Allows users to end their turn.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.11.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to end their turn.

9.11.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.11.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.12 ConveyOnceFirstCard Module inherits Command Module

9.12.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.12.2 Outputs

Name	Value Type	Description
ConveyOnceFirstCard	*Object* Command	Initialized instance of a command object. Allows players to convey their first extra card.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.12.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to convey their first extra card in player's transmuter.

9.12.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.12.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.13 ConveyOnceSecondCard Module inherits Command Module

9.13.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.13.2 Outputs

Name	Value Type	Description
ConveyOnceSecondCard	*Object* Command	Initialized instance of a command object. Allows players to convey their second extra card.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.13.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to convey their second extra card in player's transmuter.

9.13.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.13.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.14 ConveyTwiceFirstOrder Module inherits Command Module

9.14.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.14.2 Outputs

Name	Value Type	Description
ConveyTwiceFirstOrder	*Object* Command	Initialized instance of a command object. Allows players to convey both of their extra cards with the first card going in first.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.14.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to convey both of their extra cards with the first card going in first in player's transmuter.

9.14.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.14.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.15 ConveyTwiceSecondOrder Module inherits Command Module

9.15.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.15.2 Outputs

Name	Value Type	Description
ConveyTwiceSecondOrder	*Object* Command	Initialized instance of a command object. Allows players to convey both of their extra cards with the second card going in first.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.15.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to convey both of their extra cards with the second card going in first in player's transmuter.

9.15.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.15.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.16 FillMonumentWithConstructiveEnergy Module inherits Command Module

9.16.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.16.2 Outputs

Name	Value Type	Description
FillMonumentWithConstructiveEnergy	*Object* Command	Initialized instance of a command object. Allows players to fill the first available monument wall section with constructive energy type.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.16.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to fill the first available monument wall section with constructive energy type. If the player does not have the energy type available in his hand or the section is already filled or not accepting the particular energy type, check function would already catch that and won't allow player to execute this particular action.

9.16.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.16.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.17 FillMonumentWithInvertibleEnergy Module inherits Command Module

9.17.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.17.2 Outputs

Name	Value Type	Description
FillMonumentWithInvertibleEnergy	*Object* Command	Initialized instance of a command object. Allows players to fill the first available monument wall section with invertible energy type.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.17.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to fill the first available monument wall section with invertible energy type. If the player does not have the energy type available in his hand or the section is already filled or not accepting the particular energy type, check function would already catch that and won't allow player to execute this particular action.

9.17.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.17.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.18 FillMonumentWithGenerativeEnergy Module inherits Command Module

9.18.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.18.2 Outputs

Name	Value Type	Description
FillMonumentWithGenerativeEnergy	*Object* Command	Initialized instance of a command object. Allows players to fill the first available monument wall section with generative energy type.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.18.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to fill the first available monument wall section with generative energy type. If the player does not have the energy type available in his hand or the section is already filled or not accepting the particular energy type, check function would already catch that and won't allow player to execute this particular action.

9.18.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.18.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.19 FillMonumentWithPrimalEnergy Module inherits Command Module

9.19.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.19.2 Outputs

Name	Value Type	Description
FillMonumentWithPrimalEnergy	*Object* Command	Initialized instance of a command object. Allows players to fill the first available monument wall section with primal energy type.
Execute	None	Executes the action when the AI agent selects this action from the action list.
Check	Boolean	Checks whether the action is a legal action for the current player given the game state.

9.19.3 Description of Behaviour

This module is a child class of Command class. It allows the current player to fill the first available monument wall section with primal energy type. If the player does not have the energy type available in his hand or the section is already filled or not accepting the particular energy type, check function would already catch that and won't allow player to execute this particular action.

9.19.4 Exception Handling

Exception handling is done by the check function. If the move is illegal, AI agent will receive False and won't be able to execute the action in the first place.

9.19.5 Initialization

Command class and all of its sub-classes are given arguments of player and engine and automatically initialized in the ActionInitiater module (see ActionInitiater module for more details).

9.20 ActionInitiater Module inherits Command Module

9.20.1 Inputs

Name	Value Type	Description
Player	Player	Player object which will use the actions that inherits command class.
Engine	Engine	Engine object which models the game board and game rules.

9.20.2 Outputs

Name	Value Type	Description
Get_Actions	List$\langle Command \rangle$	Initializes all the subclasses of Command module and returns it as a list.

9.20.3 Description of Behaviour

This module only have one function, it is static and called in the Engine module. It goes through all the modules in the src/env/actions package (subfolder) and initializes all the subclasses of the Command module and returns it as a list of actions to be used in the game engine and game environment to allow AI Agents to select useful actions by observing the action space.

9.20.4 Exception Handling

Exception handling is done by the python's __init__.py file which makes the actions folder a package where if it cannot find a file will throw an error. There is no other special error handling done by our system.

9.20.5 Initialization

No initialization is needed. ActionInitiater only have one function, it is static and called in the Engine module.

9.21 JSON Module

9.21.1 Inputs

Name	Value Type	Description
Agent	String	A string representing the AI Agent that played the corresponding action.
Turn_Num	$\mathbb{N}$	A numerical value represented the turn that the action was played on.
Action	$\mathbb{N}$	An array index that corresponds to the action that AI Agent will play.
Action_Details	String	Additional details about the specified action.
Current_Score	Float	The current score that the agent has after playing the action.

9.21.2 Outputs

Name	Value Type	Description
Simulation_History	JSON File	Contains a JSON of each action taken during a single game simulation along with the player, turn number and action number.

9.21.3 Description of Behaviour

This module contains the attributes and methods used to gather the information from the simulation and then store them in a JSON file for further use in other modules.

9.21.4 Exception Handling

Exception handling will be done during the reading and writing of the JSON file to validate that the correct information is being written to the right file, and no extraneous files are being created during this process.

9.21.5 Initialization

The simulation history file will be initialized at the start of the simulation, in which the history be constantly updated as each move is made by the AI agents in the simulation.

9.22 JSONDataParser Module

9.22.1 Inputs

Name	Value Type	Description
jsonFile	String	Path of the JSON File

9.22.2 Outputs

Name	Value Type	Description
jsonData	JSON Object	JSON data object fetched from JSON File

9.22.3 Description of Behaviour

This module is the bridge between Game Engine and Data Visualization model. It will effectively help the Data Visualization model fetch data from the JSON log file in Game Engine.

9.22.4 Exception Handling

Exception handling will be done during the initialization of the JSONDataParser to validate the jsonFile. Errors will be thrown if the jsonFile path is invalid or the content in the jsonFile is not in JSON format.

9.22.5 Initialization

A JSONDataParser will be initialized after the JSON log file is generated from the game engine.

9.23 PieChart Module

9.23.1 Inputs

Name	Value Type	Description
jsonInput	JSON Object	JSON data fetched from the log file
edgecolor	String	Edge color that separates different wedges of the piechart
chartTitle	String	Title of the pie chart
pngTargetDirectory	String	The file path of the expected pie chart png output
objectName	String	The object that uses for pie chart statistic calculation

9.23.2 Outputs

Name	Value Type	Description
pieChartPNG	png file	Pie Chart model in png file format

9.23.3 Description of Behaviour

This module contains attributes and methods that will be used to define the statistical information of the Pie Chart. It has methods to retrieve, change, and assign the statistic input and the characteristics of the Pie Chart. After the PieChart object is initialized, it will generate a pie Chart model in the target directory.

9.23.4 Exception Handling

The Exception handling will be done during the initialization of PieChart to validate the input. Errors will be thrown if the statistic objectName and pngTargetDirectory are invalid.

9.23.5 Initialization

A PieChart object will be initialized after inputting appropriate jsonInput, objectName, and pngTargetDirectory.

9.24 BarChart Module

9.24.1 Inputs

Name	Value Type	Description
jsonInput	JSON Object	JSON data fetched from the log file
chartTitle	String	Title of the bar chart
pngTargetDirectory	String	The file path of the expected bar chart png output
objectName	String	The object that uses for bar chart statistic calculation

9.24.2 Outputs

Name	Value Type	Description
BarChartPNG	png file	Bar Chart model in png file format

9.24.3 Description of Behaviour

This module contains attributes and methods that will be used to define the statistical information of the Bar Chart. It has methods to retrieve, change, and assign the statistic input and display the distribution of data points in the Bar Chart. After the BarChart object is initialized, it will generate a horizontal bar chart model in the target directory.

9.24.4 Exception Handling

The Exception handling will be done during the initialization of BarChart to validate the input. Errors will be thrown if the statistic objectName and pngTargetDirectory are invalid.

9.24.5 Initialization

A BarChart object will be initialized after inputting the appropriate jsonInput, objectName, and pngTargetDirectory.

9.25 LineChart Module

9.25.1 Inputs

Name	Value Type	Description
jsonInput	JSON Object	JSON data fetched from the log file
chartTitle	String	Title of the line chart
pngTargetDirectory	String	The file path of the expected bar chart png output
targetDataType	String	target Object to compare
Players	List(Players)	The list of Player objects

9.25.2 Outputs

Name	Value Type	Description
LineChartPNG	png file	Line Chart model in png file format

9.25.3 Description of Behaviour

This module contains attributes and methods that will be used to display the graphic information in the Line Chart. It has methods to retrieve, change, and assign the data points, show their trends and efficiently make a comparison within the Line Chart. After the LineChart object is initialized, it will generate a Line Chart model in the target directory.

9.25.4 Exception Handling

The Exception handling will be done during the initialization of LineChart to validate the input. Errors will be thrown if the targetDataType, pngTargetDirectory, jsonInput, Players are invalid.

9.25.5 Initialization

A LineChart object will be initialized after inputting the appropriate jsonInput, targetDataType, Players, pngTargetDirectory.

Appendix A

Limitations of Design

The high modularity of the design, each specific component is its own module and they communicate through a specified interface allows for each module to be developed independently. Though the limitation of this is one cannot fully test their module without integration with the other modules, specifically the AI Agent Module cannot be fully tested and approved as working until the Game Engine Modules and Game Environment Modules are completed. This leads to integration errors that take significant development time to solve. For example if a new game mechanic wants to be added, it is first added in the Game Engine Modules, this can be tested only within the game engine without any AI interaction. Only once the AI Agent and Game Engine modules are both updated can they be integrated and checked for bugs and errors, which occur mostly during runtime as the AI can explore many different edge cases which may not have been tested and covered in the initial testing of the Game Engine modules. Another limitation of this system is the architecture can only support turn based games as it can only accept one action for one player at a time, this means for a parallel (real-time) game the architecture will not be supported.

Benefits and Trade offs of Other Solutions

Our group found that there are not any other solutions available for the system that we are developing. While standalone application of parts of our system have been created before, no one has incorporated our 3 major components of an AI, game engine and data visualization before. For example, a standalone AI game player exists like AlphaGO. Developed by Google, it is the first computer program to defeat a professional human Go player, the first to defeat a Go world champion, and is arguably the strongest Go player in history. Our system does not focus on creating the best AI player that could beat any human playing Age Contrived but rather it is an essential part in simulating multiple iterations of the game with different strategies. We found one system that incorpates the AI and game engine but lacks the data visualization that we are aiming for. Blockus Game Solver, https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1305&context=cpesp, is quite similar to our system as it aims to find winning strategies and loopholes in a multiplayer game. The report outlines strategies that they found had better outcomes, however, it lacks in providing the reader with an easy and intuitive way to visualize the outcomes and does not provide the level of modularity that our system is being developed with.

Development Timeline

Below is an estimate for module completion start and end date, the resource will be in charge of main development, but all developers will aid in architecture and integration.

Module	Start Date	Completion Date	Resource	Description
AI Agent	11-7-2022	12-11-2022	Michael	Apart of initial MVP to setup the architecture, components will be highly coupled to get initial framework
Game Environment	11-7-2022	12-11-2022	Michael	Apart of initial MVP to setup the architecture, components will be highly coupled to get initial framework
Game Engine	11-7-2022	12-11-2022	Michael	Apart of initial MVP to setup the architecture, components will be highly coupled to get initial framework
Transmuter	12-12-2022	12-31-2022	Hargun	Core game mechanic
Player	12-12-2022	12-31-2022	Jonah	Core game mechanic
Actions (Also named Commands)	1-1-2023	1-22-2023	Jonah	Decoupling to add extensibility to additional game mechanics (Architecture change)
Monument	1-1-2023	1-22-2023	Hargun	Core game mechanic
Energy	1-1-2023	1-22-2023	Jonah	Decoupling to add extensibility to additional game mechanics (core mechanic)
JSON	1-1-2023	1-15-2023	Jeffrey	Needed to bridge simulation to data visualizer
JSONDataParser	1-1-2023	1-15-2023	Tianzheng	Need to fetch the output log data to data visualizer
PieChart	1-1-2023	1-29-2023	Tianzheng	Visualize data to solve business goals, can be done in parrallel with other modules
BarChart	1-1-2023	1-29-2023	Tianzheng	Visualize data to solve business goals, can be done in parallel with other modules
LineChart	1-1-2023	1-29-2023	Tianzheng	Visualize data to solve business goals, can be done in parallel with other modules

Table 3: Module Timeline