**State of game when it was forked from master** [**ygoduelistharry**](https://github.com/ygoduelistharry)**/**[**7-wonders-duel**](https://github.com/ygoduelistharry/7-wonders-duel)

* Included age card layout of game
* Included list of all playable cards with type and cost
* Included .py file which contains a basic version of the game and can be run in command line with player input switching between 2 players
* When playing the game the layout of the current age is displayed with available cards colour coded and not available cards hidden
* A picture containing timeline

  Description automatically generated
* Players can construct cards which are added to their board, discard cards for coins, or quit the game
* Cards which are constructed are moved to the specific player board
* When the last card of the last age is selected, the game ends but no winner is selected or displayed
* Includes 6 Class functions which make up the main part of the game
* Class Game:
  + Initially defines a single instance of the game and requests input from player
  + Main game loop - Function to select card on board and perform the appropriate action
    - Get player, opponent and age variables
    - Prompts the player to select a card
    - Prints whether card has already been chosen, whether it is covered, whether resources are missing, or whether the input is not a valid action
    - Checks whether no cards are left -> progress age
    - Otherwise updates board with input from player and changes player turn
    - Ends the game when the last age is over
    - Requests new input after input was accepted
    - Created empty functions for whether a card is constructable, and which moves are valid
    - Displays the game state i.e. current player turn, current state and cards on the board, the city/board of each player
* Class Card:
  + Define a single card. Attributes match the .csv headers
  + Sets the variables for the card which are filled from the CSV file later on
* Class CardSlot:
  + Define a card slot on board to represent selectability, visibility, etc.
  + Display the Card back to the player as either Hidden or the card based on game state and whether the card is covered or not
* Class Player:
  + Define a class for play to track tableau cards, money, etc.
  + Creates and sets initial variable for players such as coins, victory\_points, clay, etc.
  + Creates a function to construct cards (empty so far)
* Class StateVariables:
  + Randomly selected the first player if none specified
  + Starts at age 0 and military track 0
  + Changes current player turn
  + Changes the age when all cards are used up
* Class Age:
  + Reads the age layout and card list CSV’s
  + Takes dataframe of all cards and creates list of card objects representing the board for a given age
  + Updates a slot when a card is selected
  + Updates the whole board when an age is over
  + Prints visual representation of cards remaining on the board for this age

**Additional features needed**

* Select card differently?
  + card\_constructable function to check whether card is constructable given state and cost
  + construct\_card function to pay resources, add card to board, gain benefit
  + update function to update players clay, wood, etc. when card is added
  + valid\_moves function returns list of valid moves for current player
  + Create military board
  + Create science board
  + Account for military victory
  + Account for scientific victory
  + Create Wonders cards -> draft, usage, limitations
  + If go again wonders is chosen -> turn to same player
  + When game end -> Count victory points and display winner
  + Create interface for AI -> no need for player input
  + Create a playable interface
  + Only 7 out of 8 wonders can be constructed -> last one discarded
  + If victory points tied, player with most blue cards winds, if that tied as well, both players share victory (draw)

**Rules to keep in mind**

* Buying a resource costs 2 coins plus the number of resources your opponent has of that type -> you only get that resource for 1 turn, there is no limit in amount of resources to buy, thus separate action for buying needed (not nested in constructing a card)
  + If you have a yellow training card with a resource and 1 coin next to it, then buying that resource only costs 1 coin no matter how many resources of that type you opponent produces (see below)
  + 
  + If multiple resources are listed as options, then the player can choose which of the resources to produce at each turn
* Constructing a card which has a symbol as prerequisite and you own the card with the respective symbol allows you to build that card for free -> (no material/coin costs)
* Military points -> for each military point move the conflict one step in the direction of the opponent (if a dotted line is crossed -> apply the effect now (lose coins) and remove the military token)
  + If the conflict pawn is moved all the way to the opponents side -> you immediately win
  + Player with weakest military chooses who begins the new age (if its in the centre, the player who played a card last will play first)
* Science -> any time you build a science structure that provides you with an identical pair of symbols, choose one of the progress tokens from the gameboard to keep (unique benefits)
  + If you have ownership of six different science symbols, you immediately win the game
* Wonders (8 wonders randomly selected) -> at the beginning of the game 4 wonders are placed in the middle, a random player starts and chooses 1 wonder, the next player chooses 2 and the first player chooses 1 again, for the next four player 2 starts in the same fashion
  + Only 7 wonders can be build -> the eighth wonder is discarded
  + To build a card you can place it under the wonder and pay the cost of the wonder instead of the cost of the card
* Victory points
  + Victory points awarded by Guild cards for each e.g. red card are awarded and counted up at the end of the game -> count towards all red cards that the player has at the end of the game not at the current time when it is played (one victory point for each red card in the city with the most red cards)

**Additional features added**

* Replaced pandas dataframe with numpy arrays
* Configured card\_constructable function to only allow constructing cards when enough money or resources are available
* Configured construct\_card to decrease coins when card is constructed
* Configured interface using s + card number to display an image of the card
* Configured interface to display whole rows with “s” -> Switch between them with W and S
* Added card effects for all cards except Yellow, Green, Purple
* Added indicator if card is selectable in GUI
* Added card effect for green cards
* Partially added card effects for purple cards (except for V points at end)
* Added a function to display player board beneath the row function using show
* Partially Handle yellow card effects (except for variable production and trading rates)
* Implemented automatic trading (automatic buying of resources if not enough resources owned)
* Handle yellow card fixing trading rates for some resources to 1
* Handle yellow card variable production (always select resource with highest trading rate which is replaced with variable production)
* Added free construction when card has a prerequisite
* Added functionality to display player boards below whole Age board and switch with “w” and “s”
* Visually implemented military board with moving conflict pawn based on military points
* Implemented military victory
* Implemented command line military board indicator
* Implement other military board functionalities (military tokens i.e. visual and coin function)
* Implemented victory points of conflict pawn
* Fixed 3 cards which were wrong in excels
* Fixed issue with green cards not giving victory points
* Visual coin indicator
* Handle civilian Victory:
  + Created civilian victory at end of game
  + Handle victory points for each set of 3 coins
  + In case of tie -> player with most blue buildings, if that ties as well -> draw
  + Handle victory points for purple cards at end of game
  + Handle victory points from Wonders
  + Handle victory points from Progress Tokens
* Create scientific victory
* Look into turning over turn when age is over -> player with weakest military chooses (or last player when military\_track = 0)
* Create progress tokens, token slots on board, randomly drafting 5, token slots of players, select token functionality if 2 of same science symbol owned, create excel representing the effects of science tokens, command line indicator of tokens?, command line indicator of owned tokens
* Visually display tokens on board based on availability
* Visual indicator of owned tokens
* Created all token effects (except Architecture and Theology)
* Stack player cards based on color to make representation more compact
* Created visual representation of wonder drafting and wonders owned/constructed during play
* Partially added all wonder functionality (except wonder\_constructable and construct\_wonder)
* Added almost all wonder effects
* Add wonder effect of The Great Library & The Mausoleum
* Valid\_moves function: Whenever player input is requested and player.type == AI, generate all valid moves the AI can perform based on the current state and which type of action needs to be performed e.g. constructing cards or choosing tokens (6 different times the input() function is called -> so 6 different valid\_move sets should be generated depending on the situation)
* API for AI agent: Whenever player input is requested (not only main loop but also e.g. progress token selection) -> if player.type == Human -> normal, elif player.type == AI -> call getAction(current state values, valid\_moves) of agent to select an action
* Finished GreedyCivilianAgent
* Create Military Greedy Agent
* Create Scientific Greedy Agent
* Created RuleBasedAgent
* Make code much more efficient (reduced time taken by 70%)
* Setup Learning environment (moved running multiple games into the game class, agent initialization only once at beginning, created final function)
* Created framework for simple reinforcement learning agent
* Create deep copy of state (shallow .copy() doesn’t work because of custom classes) -> need to create a copy method for every class
* Adjusted meaningfulness of states and actions (converted actions into their actual name e.g. choose The Sphinx instead of w1 and created a unique state key)
* Adjusted simple qLearning agent which updates qValues at the end of the game based on win or loss
* Difficulties specifying action and state keys for q-learning (too many possibilities leads to unique keys and no learning)
* Adjusted state and action key to reduce observation and action space
  + Action key: summarized discard and destroy action,
  + State key: shortened state through Booleans if player has more victory points, military points, collected science symbol, summed resources and yellow card count
* Added way to load and save trained agents
* Tried out all cards list with Boolean if player owns that card -> state space enormous (> 2,000,000 values)
* Optimized hyperparameters alpha, epsilon, gamma, MaxAttempts by running 8 agents in parallel with varying number of games and combinations (number of games varied from 10,000 to 300,000 and opponent was chosen from one of RandomAgent, RuleBasedAgent, GreddyCivilianAgent, GreedyMilitaryAgent, GreedyScientificAgent)
* Was able to improve the simple qLearning agent to perform almost comparable to the RuleBasedAgent (see below). However, not possible to improve agent further.
  + RL\_Q\_200k\_0.05\_0.02\_0.65\_2 (taining episodes, alpha, epsilon, gamma, maxAttempts)
  + 040 (RandomAgent) / 960 (LearningAgent)
  + 582 (RuleBasedAgent) / 418 (LearningAgent)
  + 187 (GreedyCivilianAgent) / 812 (LearningAgent)
  + 352 (GreedyMilitaryAgent) / 648 (LearningAgent)
  + 230 (GreedyScientificAgent) / 770 (LearningAgent)
* Completely re-configured the game environment to adhere to the OpenAI Baselines3 gym environment standards to make it compatible with standardized agents and make it more easily accessible and usable for others wanting to train their own agents
* Set-up train and test environment of maskable PPO agent
* Converted baseline agents into new format
* Trained various agents
* Train various agents
  + ~~Try with full action space 1065 instead of 273~~
  + ~~Train for longer e.g. 500,000~~
  + ~~Train against RuleBasedAgent~~
  + ~~Test Random trained against all others~~
  + ~~Test RuleBased trained against all others~~
* Illegal actions can also occur during testing (using predict and step) -> penalize them with large negative reward or change so agent is simply asked again

**Challenges encountered**

* Difficulty of current RL solutions to consider action spaces with varying sizes (at the moment invalid actions are usually rewarded with a large negative utility but convergence can be very slow until the agent realizes which actions not to take)
  + Game has a very large action space with 1065 different actions in each stage out of which not all are valid

**Additional features needed**

* Another contribution could be by comparing the performance of an agent when training using a valid\_action\_mask to only select legal actions compared to an agent which can take any action but is penalized for it if they are illegal (convergence should be much better) <https://doi.org/10.48550/arXiv.2006.14171>
* Play against MCTS
* Test MCTS 100 games against RuleBased
* Test MCTS 100 games against PPO\_15M\_Random\_RuleBased\_Agent
* Train PPO\_15M\_Random\_RuleBased\_Agent for another 5M steps?
* Play against Train PPO\_15M\_Random\_RuleBased\_Agent or newest agent

PPO MaskableMultiInputActorCriticPolicy 200k training against RandomAgent

* Wins Player 1: 32/1000 (RandomAgent)
* Wins Player 2: 968/1000 (PPO\_200k\_Random\_Agent)

RuleBasedAgent against RandomAgent

* Wins Player 1: 33/1000 (RandomAgent)
* Wins Player 2: 967/1000 (RuleBasedAgent)

PPO\_200k\_Random against RuleBasedAgent

* Wins Player 1: 719/1000 (RuleBasedAgent)
* Wins Player 2: 281/1000 (PPO\_200k\_Random\_Agent)

PPO\_500k\_Random against RandomAgent

* Wins Player 1: 10/1000 (RandomAgent)
* Wins Player 2: 990/1000 (PPO\_500k\_Random\_Agent)

PPO\_500k\_Random against RuleBasedAgent

* Wins Player 1: 478/1000 (RuleBasedAgent)
* Wins Player 2: 522/1000 (PPO\_500k\_Random\_Agent)
* 010 (RandomAgent) / 990 (PPO\_500k\_Random)
* 478 (RuleBasedAgent) / 522 (PPO\_500k\_Random)
* 017 (GreedyCivilianAgent) / 983 (PPO\_500k\_Random)
* 030 (GreedyMilitaryAgent) / 970 (PPO\_500k\_Random)
* 037 (GreedyScientificAgent) / 963 (PPO\_500k\_Random)

PPO\_2000k\_RuleBased

* Wins Agent 1: 233/1000 (RuleBasedAgent)
* Wins Agent 2: 767/1000 (PPO\_2000k\_RuleBased\_Agent)
* 004 (RandomAgent) / 996 (PPO\_2000k\_RuleBased)
* 233 (RuleBasedAgent) / 767 (PPO\_2000k\_RuleBased)
* 025 (GreedyCivilianAgent) / 975 (PPO\_2000k\_RuleBased)
* 115 (GreedyMilitaryAgent) / 885 (PPO\_2000k\_RuleBased)
* 061 (GreedyScientificAgent) / 939 (PPO\_2000k\_RuleBased)

PPO\_10000k\_Random\_RuleBased

* 002 (RandomAgent) / 998 (PPO\_10000k\_Random\_RuleBased)
* 194 (RuleBasedAgent) / 806 (PPO\_10000k\_Random\_RuleBased)
* 014 (GreedyCivilianAgent) / 986 (PPO\_10000k\_Random\_RuleBased)
* 027 (GreedyMilitaryAgent) / 973 (PPO\_10000k\_Random\_RuleBased)
* 021 (GreedyScientificAgent) / 979 (PPO\_10000k\_Random\_RuleBased)

PPO\_15M\_Random\_RuleBased\_Agent

* 003 (RandomAgent) / 997 (PPO\_15M\_Random\_RuleBased\_Agent)
* 128 (RuleBasedAgent) / 872 (PPO\_15M\_Random\_RuleBased\_Agent)
* 010 (GreedyCivilianAgent) / 990 (PPO\_15M\_Random\_RuleBased\_Agent)
* 007 (GreedyMilitaryAgent) / 993 (PPO\_15M\_Random\_RuleBased\_Agent)
* 030 (GreedyScientificAgent) / 970 (PPO\_15M\_Random\_RuleBased\_Agent)
* 377 (PPO\_10000k\_Random\_RuleBased) / 622 (PPO\_15M\_Random\_RuleBased\_Agent)
* 430 (PPO\_10000k\_newScore) / 570 (PPO\_15M\_Random\_RuleBased\_Agent)
* 513 (PPO\_10M\_vs\_PPO\_7\_RuleBased\_3) / 487 (PPO\_15M\_Random\_RuleBased\_Agent)
* 497 (PPO\_10M\_vs\_PPO\_5\_RuleBased\_5) / 503 (PPO\_15M\_Random\_RuleBased\_Agent)

General Tests

* 224/776 🡪 RandomAgent vs. GreedyCivilianAgent
* 227/773 🡪 RandomAgent vs. GreedyMilitaryAgent
* 426/573 🡪 RandomAgent vs. GreedyScientificAgent
* 233/766 🡪 GreedyCivilianAgent vs. GreedyMilitaryAgent
* 450/550 🡪 GreedyCivilianAgent vs. GreedyScientificAgent
* 796/204 🡪 GreedyMilitaryAgent vs. GreedyScientificAgent
* 033/967 🡪 RandomAgent vs. RuleBasedAgent
* 915/085 🡪 RuleBasedAgent vs. GreedyCivilianAgent
* 880/120 🡪 RuleBasedAgent vs. GreedyMilitaryAgent
* 961/039 🡪 RuleBasedAgent vs. GreedyScientificAgent

Testing:

* dict\_size ~80,000 (more military)
* Wins Player 1: 95/1000 (RandomAgent)
* Wins Player 2: 905/1000 (LearningAgent)
* Draws: 0/1000
* After 40,000 training episodes
* dict\_size ~110,000 (more military)
* Wins Player 1: 162/1000 (RandomAgent)
* Wins Player 2: 837/1000 (LearningAgent)
* Draws: 1/1000
* After 80,000 training episodes
* 103 (RandomAgent) / 897 (LearningAgent)
* 754 (RuleBasedAgent) / 246 (LearningAgent)
* 331 (GreedyCivilianAgent) / 667 (LearningAgent)
* 438 (GreedyMilitaryAgent) / 562 (LearningAgent)
* 342 (GreedyScientificAgent) / 658 (LearningAgent)
* dict\_size ~100,000
* Wins Player 1: 753/1000 (RuleBasedAgent)
* Wins Player 2: 247/1000 (LearningAgent)
* Draws: 0/1000
* After 20,000 training episodes
* dict\_size ~140,000
* Wins Player 1: 707/1000 (RuleBasedAgent)
* Wins Player 2: 293/1000 (LearningAgent)
* Draws: 0/1000
* After 40,000 training episodes
* dict\_size ~200,000
* Wins Player 1: 688/1000 (RuleBasedAgent)
* Wins Player 2: 312/1000 (LearningAgent)
* Draws: 0/1000
* After 80,000 training episodes
* 129 (RandomAgent) / 871 (LearningAgent)
* 688 (RuleBasedAgent) / 312 (LearningAgent)
* 346 (GreedyCivilianAgent) / 654 (LearningAgent)
* 489 (GreedyMilitaryAgent) / 511 (LearningAgent)
* 248 (GreedyScientificAgent) / 752 (LearningAgent)
* dict\_size ~220,000
* Wins Player 1: 334/1000 (GreedyMilitaryAgent)
* Wins Player 2: 666/1000 (LearningAgent)
* Draws: 0/1000
* After 80,000 training RuleBased, 20,000 Military
* **Performance against all other agents much worse**
* RL\_Q\_200k
* 110 (RandomAgent) / 890 (LearningAgent)
* 613 (RuleBasedAgent) / 387 (LearningAgent)
* 289 (GreedyCivilianAgent) / 711 (LearningAgent)
* 402 (GreedyMilitaryAgent) / 598 (LearningAgent)
* 200 (GreedyScientificAgent) / 799 (LearningAgent)
* Hyperparameters: 50k training
* [Alpha, Gamma, maxAttempts] = [0.05, 0.02, 0.7, 2]
* Wins Player 1: 639/1000 (RuleBasedAgent)
* Wins Player 2: 360/1000 (LearningAgent)
* Draws: 1/1000
* [Alpha, Gamma, maxAttempts] = [0.05, 0.02, 0.5, 2]
* Wins Player 1: 606/1000 (RuleBasedAgent)
* Wins Player 2: 394/1000 (LearningAgent)
* Draws: 0/1000
* RL\_Q\_50k\_0.05\_0.02\_0.5\_2 = 50k training
* Wins Player 1: 6119/10000 (RuleBasedAgent)
* Wins Player 2: 3880/10000 (LearningAgent)
* Draws: 1/10000
* RL\_Q\_200k\_0.05\_0.02\_0.65\_2 = 200k training
* Wins Player 1: 5797/10000 (RuleBasedAgent)
* Wins Player 2: 4197/10000 (LearningAgent)
* Draws: 6/10000
* RL\_Q\_200k\_0.05\_0.02\_0.65\_2
* 040 (RandomAgent) / 960 (LearningAgent)
* 582 (RuleBasedAgent) / 418 (LearningAgent)
* 187 (GreedyCivilianAgent) / 812 (LearningAgent)
* 352 (GreedyMilitaryAgent) / 648 (LearningAgent)
* 230 (GreedyScientificAgent) / 770 (LearningAgent)

Statistics (Wins 1/ Draws / Wins 2)

* 2295/03/7702 🡪 RandomAgent vs. GreedyCivilianAgent
* 2991/10/6999 🡪 RandomAgent vs. GreedyMilitaryAgent
* 4785/08/5207 🡪 RandomAgent vs. GreedyScientificAgent
* 2857/00/7143 🡪 GreedyCivilianAgent vs. GreedyMilitaryAgent
* 4955/00/5045 🡪 GreedyCivilianAgent vs. GreedyScientificAgent
* 7816/03/2181 🡪 GreedyMilitaryAgent vs. GreedyScientificAgent
* 0439/01/9560 🡪 RandomAgent vs. RuleBasedAgent
* 9092/00/0908 🡪 RuleBasedAgent vs. GreedyCivilianAgent
* 8946/00/1054 🡪 RuleBasedAgent vs. GreedyMilitaryAgent
* 9638/02/0360 🡪 RuleBasedAgent vs. GreedyScientificAgent

Time:

* 1 game takes ~0.03 seconds muted or ~0.05 seconds unmuted
* Reading parameters: 0.001s
* Using different Agents does not impact the time (e.g. RandomAgent compared to RuleBasedAgent)
* Setup of game class only takes 0.002s
* Valid\_moves function seems to take a lot of time around 70% especially the 2 for loops in it calling the functions:
  + Function card\_constructable
  + Function wonder\_constructable
* 1000 games take around 26s
* Reduced time down to 9s for 1000 games
* Increase to 19s through deepcopy of current state for self.lastState