A BOARD GAME-BASED VIRTUAL ENVIRONMENT FOR INTELLIGENT BOTS PROGRAMMING

A. Heras, V. Sanchez-Anguix, J.M. Alberola, A. Perez-Pascual

Universitat Politecnica de Valencia (SPAIN)

Abstract

Nowadays, there are few virtual environments based on board games with a didactic purpose. In fact, a new board game-based environment is rarely created for training bots unless it is necessary for a study. However, the development of intelligent bots applied to such games would be a stimulus to motivate disciplines such as programming or Artificial Intelligence. In this paper, we present a virtual environment based on a well-known board game such as Catan, which allows the incorporation of bots that can play against each other. In this sense, the virtual environment allows the development of new bots with their respective own strategies and algorithms, so that simulations of games can be carried out to measure their effectiveness. In addition, it also allows the simulation of multiple games to develop bots that incorporate learning techniques based on Artificial Intelligence or Machine Learning. In this sense, the virtual environment offers a very interesting tool to be used in subjects related to these disciplines.

Keywords: Artificial Intelligence, Virtual environment, Bots programming, simulation, board-games.

1 INTRODUCTION

Simulation-based environments have been widely used in different areas such as nursing [1], maths [2], medicine [3], and English instruction [4]. These environments are usually built by using Artificial Intelligence techniques such as artificial neural networks, natural language processing, or machine learning [5]. However, it is difficult to find simulation environments for teaching Artificial Intelligence itself.

When observing courses and subjects related to Artificial Intelligence topics, such as those based on the Association for Computing Machinery (ACM) curriculum recommendations [6], it can be observed that some fields of Artificial Intelligence are widely and commonly taught, such as Machine Learning, adversarial search, or intelligent agents. We argue that simulation-based environments are very powerful to teach some of these topics since these environments allow students to develop practical abilities based on theoretical knowledge.

Regarding simulation, several simulators can be found in the literature that have been used for academic purposes such as NetLogo [7], CoppeliaSim [8] or JGomas [9]. However, it is difficult to find simulation environments for board games with a didactic purpose, as proposed in [10] for the game Diplomacy. Through these board games, students could develop intelligent bots that implement player behaviours and simulate different games to test their development.

There are different and popular board games with less or more complexity that could be interesting for teaching Artificial Intelligence techniques, such as Ciudadelas, Cluedo, or Carcassonne. However, these games lack player negotiation, which, from our point of view, reduces the complexity of what students could implement with an intelligent bot. The development of these intelligent bots applied to such games would be a stimulus to motivate disciplines such as programming or Artificial Intelligence.

As a result, this paper presents a virtual environment that allows the simulation of a popular board game such as Catan. This board game has specific features that make it more complex than a game heavily dependent on luck, such as a negotiation process among players. Therefore, the virtual environment allows the programming of intelligent bots with different decision-making capabilities using Artificial Intelligence techniques. In this way, bots with different strategies can be tested against each other. Unlike other similar works like jSettlers [11], our virtual environment facilitates the implementation of Artificial Intelligence techniques by providing an inheritable interface with which a technique can be directly integrated into the game. Additionally, our environment allows for running simulations by executing only bots without human intervention, which enables training various Artificial Intelligence techniques to define the bot's behaviour.

The rest of the paper is organized as follows. In Section 2 we describe the virtual environment, including the general architecture and the requirements for bots programming. In Section 3 we show some experiments to validate our proposal. Finally, in Section 4 we draw some concluding remarks and future works.

2 DESCRIPTION OF THE VIRTUAL ENVIRONMENT

The general architecture of the virtual environment is shown in Fig. 1. As it can be observed, there are two key components: the simulation environment and the visualizer. This separation allows the execution of multiple games that may be necessary for training learning models. The execution of a game is carried out in the simulation environment while the visualizer, which is responsible for showing the result of the game. In this visualization process, every step that has occurred in each round of the game can be analysed, so a JSON file needs to be loaded into the visualizer. The advantage of exporting the execution trace of a game to a JSON file is that this information allows to carry out an underlying learning process inside a bot, without using the visualizer.

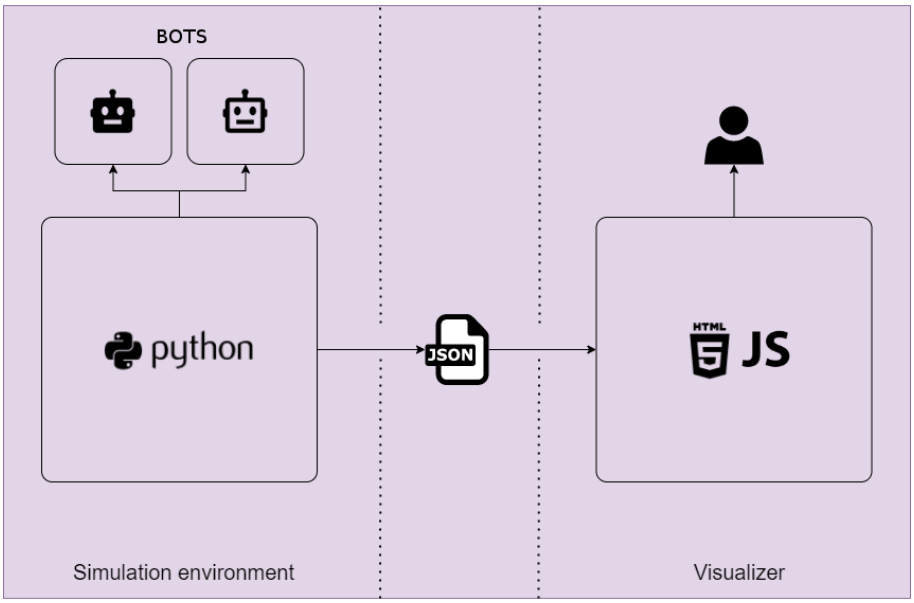

Figure 1: Virtual environment architecture

The visualizer represents the frontend component of the virtual environment. This component is responsible for displaying all relevant information about the execution of each round in a simple way (Fig. 2).

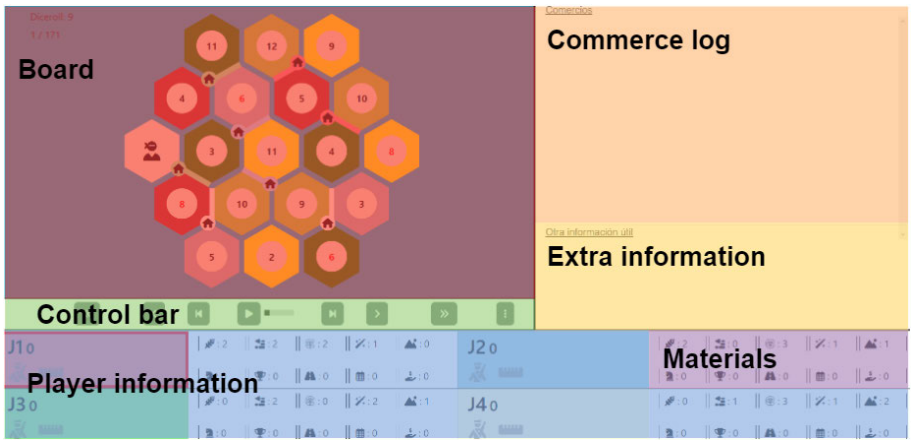

Figure 2: Scheme of the different panels of the visualizer

As can be appreciated, the design was made with the aim of being functional, showing both the information of the map and the players as well as what happens during the execution of each round. This design was also thought to allow quick visualisation without consulting a user guide. The map design has the classic hexagonal shape of Catan for positioning cities and roads. It also provides a bottom menu that allows the selection of each round of the game.

The simulation environment represents the backend component. This component is responsible for carrying out the execution of each round of the game by following the flow diagram shown in Fig. 3.

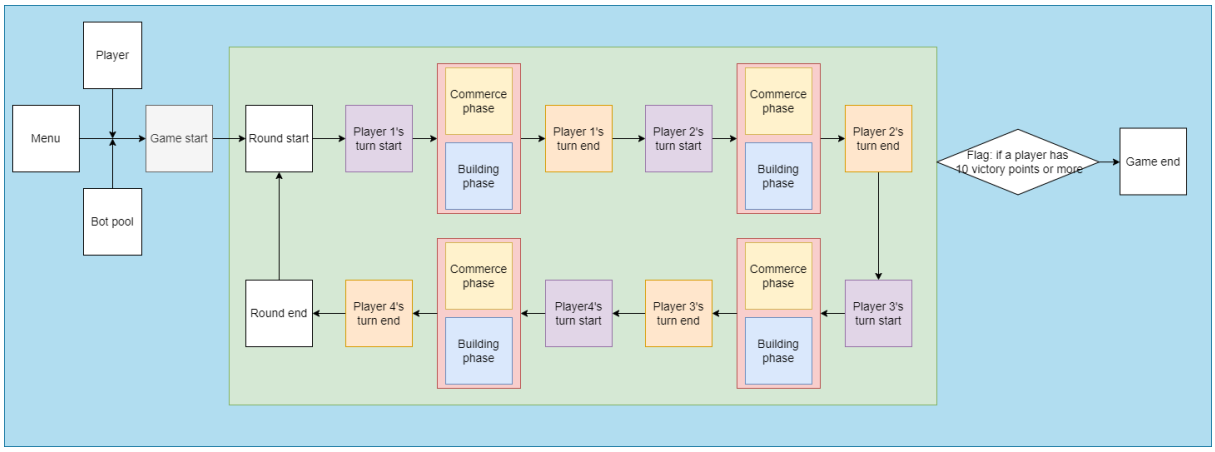

Figure 3: Flow diagram of the game

As can be seen, a game consists of a round loop, where turns are passed between the four players. This implies that each game is composed of an indeterminate number of rounds, however, all of them are similar in terms of execution possibilities. It can also be observed that the turns are similar, while each player is characterised by a number and has his/her own materials, cities, and roads.

The structure of each turn is composed of four phases: the initial or production phase where the dice is rolled and materials are received, the trading phase where trades are proposed with other players, the construction phase where materials are used for the buildings, and the final phase, which represents the turn end and where the victory points are counted, and the special condition cards are handed out if those conditions are met. In case any player obtains 10 victory points at the end phase, the game ends.

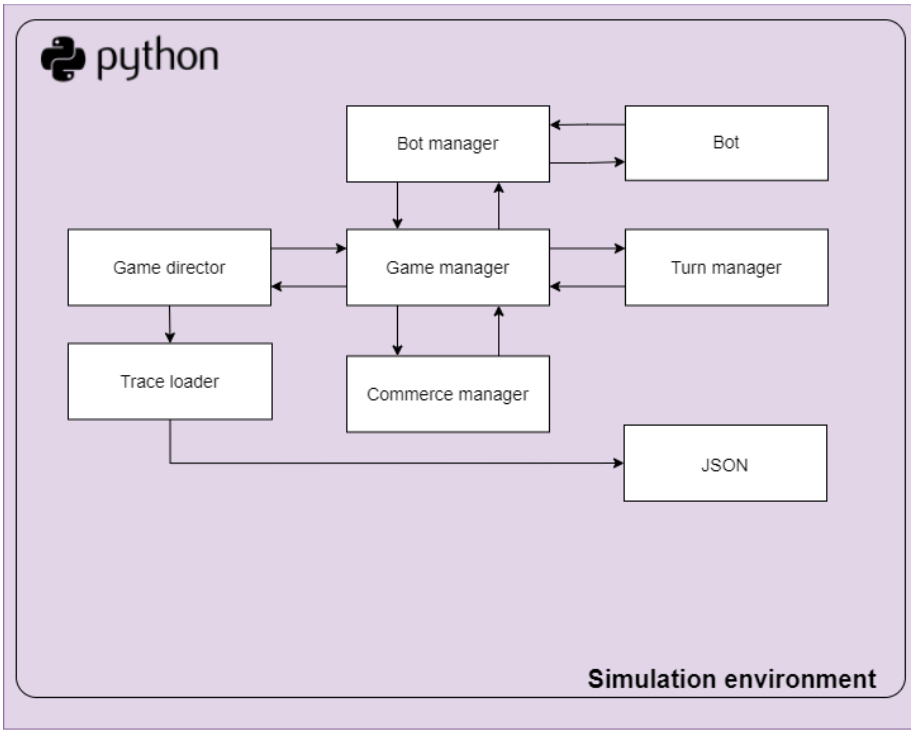

Figure 4: Architecture of the simulation environment

The internal architecture of the simulation environment is shown in Fig. 4. This architecture is composed of several components:

- **GameDirector** controls the game by deciding the turn of each player and what action to take at each moment. It also controls the order of rounds and checks if there is a winner.
- **GameManager** is responsible for executing the actions defined by the GameDirector. It also ensures that the actions taken by the bots are always allowed and prevents them from performing improper actions.
- **TurnManager** is responsible for counting the rounds, turns, and phases of the game.
- **CommerceManager** is responsible for performing trades.
- **BotManager** controls the information associated with each bot (materials, development cards, and cities).
- **TraceLoader** is responsible for exporting a trace of the game to a JSON object for the visualizer.

2.1 Bots programming

The **BotInterface** interface is the component associated with bot programming. This interface contains a set of triggers to be implemented, which define the logic of the bot. These triggers are implemented in *Python* and act according to the different events they are associated with. If these triggers are not implemented, the bot's decision-making is neutral and therefore has no intelligence. These triggers are as follows:

- **On_game_start** is called when the game starts in order to choose where to place a city and a road.
- **On_turn_start** is called at the beginning of the player's turn. Only allows to play development cards.
- **On_trade_offer** is called when a trade proposed by another player is received.
- **On_build_phase** is called when the building phase begins. Allows to build roads, cities, or development cards. Development cards are also allowed to be played.
- **On_turn_end** is called when the player's turn ends. Only allows to play development cards.
- **On_commerce_phase** is called during the commerce phase. Allows to send offers to other players or to the bank. Development cards are also allowed to be played.
- **On_moving_thief** is called to move the thief to another terrain tile. Allows to choose which terrain tile to move it to, and which adjacent player to steal a material from.
- **On_having_more_than_7_materials** is thrown when more than seven materials are held in hand and a 7 comes up on the die. The half of the cards must be discarded. Otherwise, the gamemanager will discard them randomly.
- **On_road_building_card_use** is thrown when the bot plays a *road building* card. Allows to choose which two roads to build.
- **On_monopoly_card_use** is thrown when the bot plays a *monopoly* card. Allows to choose which material is wanted to have the monopoly on.
- **On_year_of_plenty_use** is thrown when the bot plays a *year of plenty* card. Allows to choose two materials, which can be the same twice, and obtain one material of each type.

The above list of triggers allows different types of practices in the classroom. As an example, it could focus on the perspective of negotiation. In this case, the objective for students would be to implement a negotiation algorithm using the *on_commerce_phase* and *on_trade_offer* triggers. Another example could be focusing on blocking the expansion of other players. In this case, the objective for students would be to implement an algorithm to determine which player's road can be blocked by building their own road using the *on_build_phase* trigger.

3 RESULTS

To validate the virtual environment, a bot with an underlying simple heuristic has been implemented and compared to dummy bots that make random decisions. The bot with the simple heuristic is prepared to make strategic decisions that can give it a victory. Multiple triggers leave some random decisions. Thus,

the heuristic bot is designed to be only slightly better than the dummy bots. Below, the implementation of the different triggers associated with this heuristic is detailed:

- **On_game_start**: the bot chooses a node that is adjacent to a terrain piece with a probability of 6 or 8. If none are available, then it chooses one randomly from the viable options. The road is always made pointing in a random direction.
- **On_turn_start**: the bot plays the *knight* card if it has one in hand.
- **On_trade_offer**: the bot accepts an offer if it receives more materials than it offers. The bot does not propose counteroffers.
- **On_build_phase**: if the bot has the *road building* card and has two possible roads to build or has the *year of plenty* card, the bot plays them. After playing development cards, if the bot has at least one village built and has the materials to build a city, it builds a city.
    - Cities are only built adjacent to terrain pieces with a probability of 4/36 or greater. If the bot does not have enough materials, does not have villages or the cities would be built in a low-probability area, it tries to build a village.
    - Like cities, villages are also limited by the probability of the terrain pieces. For villages, the goal is to find a terrain piece with a probability of 3/36 or greater.
    - If the bot cannot build a village, it builds a road. The road is always built if the final node is coastal and has a port, if there is none with those requirements, it is chosen randomly from among all possible roads.
    - If the bot cannot build a road, then it builds a development card. According to this, development cards are given low priority.
- **On_turn_end**: the bot plays the *Victory Point* card if it has one in hand.
- **On_commerce_phase**: the bot plays the *Monopoly* card if it has delivered three or more of the same material to a player in the previous trade.
    - If the bot has at least one village built and already has enough materials to make a city, the bot closes the trade phase.
    - If the bot has at least one village and does not have enough materials, it asks for the ones missing to complete a city by exchanging the same number of materials.
    - If the bot does not have any villages, it asks for the materials missing to create a village, unless it already has enough, in which case it cuts the trading phase.
- **On_moving_thief**: the bot moves the thief to one of the 4 terrain pieces with a 6 or 8. It makes sure there is at least one city of an opponent and that he does not have one on that terrain piece. If the conditions are not met, the bot lets the gamemanager decide where to place it.
- **On_having_more_than_7_materials**: the bot discards materials randomly but keeps enough to build a city in case it can hold on to them. Otherwise, the bot lets the gamemanager discard them.
- **On_monopoly_card_use**: the bot chooses the material that has been traded at least three times.
- **On_road_building_card_use**: the bot chooses two random roads from the available options to make roads.
- **On_year_of_plenty_card_use**: the bot chooses the materials which are demanded when using the *year of plenty* card.

To validate the performance of the bot, 10 games were run where the heuristic bot was player 1 while the rest of the players were dummy bots (Table 1). In addition, to avoid possible advantages that may come from always being the first player, another 10 games were run where the heuristic bot was player 4 while the rest of the players were dummy bots (Table 2).

Table 1: Execution of games where player 1 runs the heuristic bot (largest army: ♞; longest road: 🛣)

	P1	♞	🛣	P2	♞	🛣	P3	♞	🛣	P4	♞	🛣
Game 1	10	✓	✓	2			4			3		
Game 2	10		✓	2			3			2		
Game 3	6	✓		10		✓	4			6		
Game 4	10	✓	✓	2			2			2		
Game 5	10		✓	2			3			2		
Game 6	10	✓		4			4			4		
Game 7	8			10	✓		6			4		
Game 8	10	✓	✓	2			2			3		
Game 9	4			3			10			5	✓	
Game 10	5			4			10			6	✓	

As it can be observed, the heuristic bot was able to win the majority of games in both cases, also obtaining the special cards of the largest army or the longest road. It is observed that the heuristic bot won 60% of the games. In this sense, it has been proven that the virtual environment allows the development of bots that, although they consist on a simple heuristic, are capable of winning against dummy bots that make random decisions in a game that is less dependent on luck, such as Catan.

Table 2: Execution of games where player 4 runs the heuristic bot (largest army: ♞; longest road: 🛣)

	P1	♞	🛣	P2	♞	🛣	P3	♞	🛣	P4	♞	🛣
Game 1	2			3			4			10	✓	
Game 2	4			4			4			10	✓	
Game 3	10	✓		4			4			4		
Game 4	5	✓		4			10			6		
Game 5	4			6			4			10	✓	
Game 6	2			10	✓		2			4		
Game 7	2			2			2			10	✓	
Game 8	2			2			2			10		✓
Game 9	3			10			3			8	✓	
Game 10	2			3			4			10	✓	

4 CONCLUSIONS

In this paper, we presented a virtual environment that allows the simulation of Catan, which is a well-known board game. This virtual environment can be used in university educational courses to support the learning process of students when practising skills related to programming and Artificial Intelligence related skills and knowledge. To do this, the virtual environment provides an easy way for implementing bots with specific strategies by using event-related triggers. Decoupling the simulation environment and the visualizer allows running hundreds and thousands of games to train various learning strategies in a shorter amount of time.

Unlike simpler board games, Catan offers a challenging environment where the player must know how to adapt to luck while trying to obtain the best trades with other players. Thanks to all this, the negotiation

component provided by Catan allows the implementation of different negotiation strategies that can be applied within an environment where a bad trade could make to lose the game.

We have tested this environment by implementing a simple heuristic-based bot and running several games against dummy bots. As we have shown in the evaluation, a short component of intelligence is enough to win the majority of the games against these dummy bots. As a future work, we plan to test this environment in the classroom in order to develop more complex bots which compete against each other. In addition, by using the visualizer to check the weaknesses of the bot, a much intelligent bot could be developed, capable of winning a greater percentage of times.

REFERENCES

[1] Lavoie, P., & Clarke, S. P. (2017). Simulation in nursing education. Nursing management, 48(2), 16-17.

[2] Gazdula, J., & Farr, R. (2020). Teaching risk and probability: Building the Monopoly® board game into a probability simulator. Management Teaching Review, 5(2), 133-143.

[3] Pottle, J. (2019). Virtual reality and the transformation of medical education. Future healthcare journal, 6(3), 181.

[4] Angelini, M. L., & García-Carbonell, A. (2019). Developing English speaking skills through simulation-based instruction. Teaching English with Technology, 19(2), 3-20.

[5] Dai, C. P., & Ke, F. (2022). Educational applications of artificial intelligence in simulation-based learning: A systematic mapping review. Computers and Education: Artificial Intelligence, 100087.

[6] ACM (2020). Computing Curricula 2020.

[7] Borowczak, M., & Burrows, A. C. (2019). Ants go marching—Integrating computer science into teacher professional development with NetLogo. Education Sciences, 9(1), 66.

[8] e Silva, R. D. A. A., Joventino, C. F., Pereira, J. H., & Correa, L. P. (2021). Teaching Robotics in Pandemic Times Through Remote Education. In 2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE) (pp. 371-376). IEEE.

[9] Hernandez, L., Esparcia, S., Julian, V., & Carrascosa, C. (2016). JGOMAS 2.0: A Capture-the-Flag Game Using Jason Agents and Human Interaction. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 173-184). Springer, Cham.

[10] Kramár, J., Eccles, T., Gemp, I., Tacchetti, A., McKee, K. R., Malinowski, M., ... & Bachrach, Y. (2022). Negotiation and honesty in artificial intelligence methods for the board game of Diplomacy. Nature Communications, 13(1), 1-15.

[11] Monin, J. (2010). Java Settlers of Catan. https://nand.net/jsettlers/