Efficient Solver for Dynamic Puzzle Games: Baba Is Y'all

Yuhang Lin, Xianqing Zeng

***Abstract*—The demand for intelligent solvers for puzzle games has overgrown with the increase in depth and complexity of these games. Most traditional puzzles with static mechanisms already have many high-performance algorithms to find solutions. However, puzzles with dynamic rule systems have occurred recently and have only a few algorithms to obtain a solution. Because of the unique dynamic mechanisms, the game *Baba Is You* is chosen as the research object. Two search algorithms, RHEA and MCTS, are performed in this game. MCTS reaches better accuracy than the four baselines, BFS, DFS, Default, and Random. But RHEA performs worst in experiments on a large and overall test-level set. Besides search algorithms, reinforcement learning will be applied in this game in the future.**

***Index Terms*—Baba Is You, Agent, Dynamic Rule, RHEA, MCTS.**

I. Introduction

With the increase in depth and complexity of puzzle games, answers to these puzzles require more intelligence and better problem-solving ability. More variables and limitations in consideration make it hard to calculate and obtain correct answers manually and may take more time. Thus the solver agent arouses people's attention. It can help people find solutions quickly and help puzzle designers better understand players' behaviors during the game process, which contributes to improvements to game design and makes it more fascinating and challenging.

A. Puzzle Category

According to the types of rules, puzzles divide into two categories: puzzles with static mechanisms and dynamic mechanisms.

Most puzzle games have a set system of rules that can be followed and will never be changed at any point during play. One of the most classic examples is *Sokoban* because of its discretized, simplistic movement and consistent rule space. In this game, the player can only push rather than pull each crate to designated positions, and this push-pull mechanism will never change by players' actions or states. These straightforward rules staying consistent from level to level are called "static mechanisms".

"Dynamic mechanism" refers to dynamically changing mechanic space affected by the player's actions. These actions may cause temporary or permanent effects on the rule system. Thus rules do not have to be initially made at the start of the game and could be manipulated at any point while the player attempts to solve the level. Some examples of this type of mechanic space include resource changes, gravity changes, and time-sequential changes. One puzzle game with perspective changes is *Monument Valley* as shown in Fig.1 and Fig.2, where the player has to rotate the entire map to navigate through the level successfully. [1]

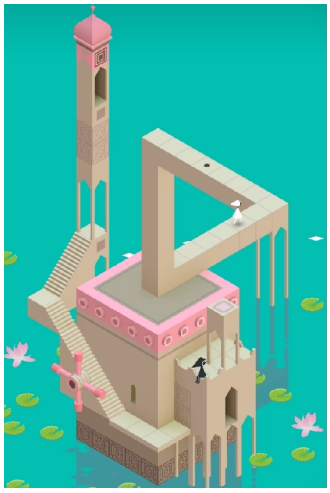

Fig. 1: Before Rotating

Fig. 2: After Rotating

B. Algorithms for puzzle games

Because of the research and study of static rules over a long history, the solution space of traditional games has been comprehensively explored, and many efficient and high-performance algorithms have already existed. This type of agent consists of the tree search algorithm, reinforcement learning, evolutionary algorithms, etc. [2]

However, since dynamic rule games have appeared in recent years, they have not been studied thoroughly. Some AI competitions exist for adaptive agents, like *Hanabi* (a multi-agent competitive strategy game) [3] and a 2D arcade-style fighting game [4]. However, the experience in such a field is still inadequate.

To allow innovative and efficient agents to emerge from this type of puzzle and provide insight into the design process and evaluation of these agents in an adaptive game environment, we choose one single game with dynamic rules as a research object and attempt to develop an intelligent agent for this game: *Baba Is Y'all*.

II. Background

Baba Is You is a puzzle game developed by Arvi Teikari originally for the 2017 Nordic Game Jam and then expanded to a full game with more mechanics. It has a similar game

style like *Sokoban*, a player can control a character in the map to push blocks into a particular space to beat the level. However, some blocks may contain words and can form rules in the game. These rules can be created or broken at any time. Much of the game involves manipulating the rules in a certain order or orientation to allow the puzzle to be solved [1]. Fig.3 shows an example.

Besides the dynamic rule system, the most significant reason for choosing this game is that there has already been some flat research about this game, which offers some experience to study.

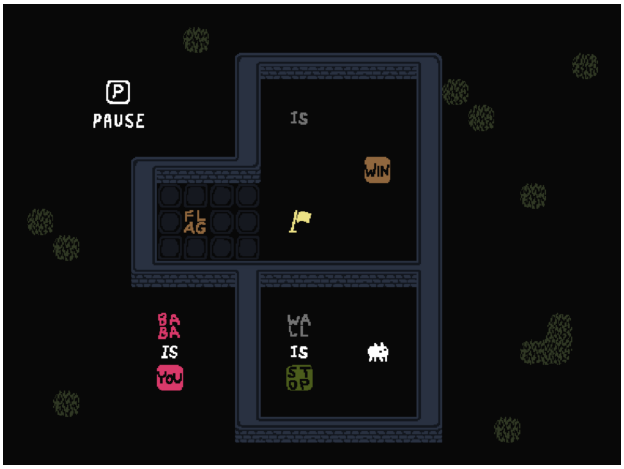

Fig. 3: A sample level in *Baba Is You*. Players have to break the active rule 'WALL-IS-STOP' and push the word blocks on the other side together, and create the rule 'FLAG-IS-WIN' to solve the level. [1]

A. *Baba is Y'all framework*

A level editor website called *Baba Is Y'all* allows players to make their levels, publish levels online, and rate levels. The high-performance levels in the website are good sources of test and debug material. We can also create our test maps to cover some extreme cases.

The integral version of *Baba Is You* contains hundreds of blocks and rules, which makes it hard to get started. This framework is the most original version instead of the latest version of the game. It only contains 11 object classes and 21 keyword classes and can form only nine formats of rules, which is a small subset of the full version of the game. Less complicated rules and core mechanisms make it the ideal material for algorithm design. [5]

B. *Keke AI Competition*

In IEEE Conference on Games 2022, a competition called "Keke AI Competition" adopts *Baba Is Y'all* framework and sets up an essential environment for programming.

The organizers reveal an open-source project containing a local website to evaluate the game [6]. The tests can fail or succeed, making it easy to verify the basic functionality of the agent and enter the improvement phase quickly. Some basic properties will also be shown on the website, like correctness rate, average time per solution, and the average length of solutions, all of which are significant figures for comparison and further improvement. One screenshot of the local evaluator is shown in Fig.4.

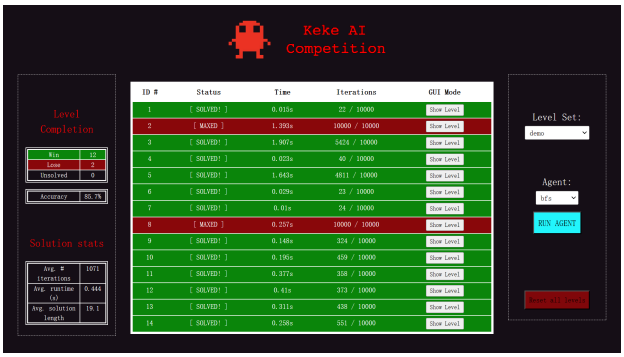

Fig. 4: The evaluator screen of the offline Keke AI Competition offline interface

The project also provides a detailed GitHub wiki with a walk-through video to illustrate general ideas and variables in the environment, which saves time in reading and understanding code. Lastly, the organizers offer some valuable baseline agents in the project, which are the only agents for *Baba Is You* online and will be explained in IV.

III. Rules for *Baba Is Y'all*

Each level of the game initially contains particular objects and keywords, and players must push them to achieve winning conditions.

A. *Object and Keyword*

"Object" refers to the former word in the format "X-IS-Y"; the object can only be placed at the former for validation. Putting an object in the last place will not form a good rule.

"Keyword" refers to the latter word in the format "X-IS-Y", and the keyword can only be placed at the latter for validation. Similarly, putting the keyword in the former place will not form a good rule. Players can push all keywords under any rules.

Table.I shows all objects and keywords.

TABLE I: Objects and Keywords

Object	Keyword
BABA	YOU
BONE	WIN
FLAG	STOP
WALL	MOVE
GRAN	PUSH
LAVA	SINK
ROCK	KILL
FLOOR	HOT
KEKE	MELT
GOOP	IS
LOVE	

B. Rule formats

Rules for the level are defined by statements readable as English text and read from up, down, and left-right. So "X-IS-YOU" would be interpreted as a valid rule, but not "YOU-IS-X". As such, all rules must take one of these three formats [2] [7]:

- **X-IS-(KEYWORD)** where KEYWORD belongs to a word in the keyword class that manipulates the property of the game object class X (i.e., 'WIN', 'YOU', 'MOVE', etc.)
- **X-IS-Y** (X≠Y) a transformative rule that changes all instances of sprites identified as X to the sprite Y.
- **X-IS-X** a reflexive rule that prevents any transformations occurring on the word block X. This differs from the previous rule X-IS-Y, as Y must differ from X for a transformation to occur. If a transformative rule is created, the X word block will not transform into Y with the reflexive tule active.

C. Detail Meaning

Table.II shows all detail keywords in the rules.

TABLE II: Objects and Keywords [1]

Rule Type	Definition
X	In this table refers to any object class
X-IS-X	objects of class X can't be changed to another class
X-IS-Y	objects of class X will transform to class Y, unless "X-IS-X" exists
X-IS-PUSH	X can be pushed
X-IS-MOVE	X will move towards its facing direction each action, X will turn around if meeting a wall. X will move whatever button the player press, even if SPACE is pressed.
X-IS-STOP	X will prevent the player from passing through it
X-IS-KILL	X will kill the player on contact
X-IS-SINK	X will erase any object contacting it, and will disappear with the contacted object
X-IS-HOT	X can only erase Y with "Y-IS-MELT", and will disappear with Y
X-IS-MELT	X can only erase Y with "Y-IS-HOT", and will disappear with Y
X-IS-YOU	All objects of X class are controlled by the player
X-IS-WIN	Player can win by touching X

D. Actions

Five actions are allowed to be performed in the game:

1) LEFT: all controlled objects move one step to the left, and all moveable objects move one step to their facing direction.
2) RIGHT: all controlled objects move one step to the right, and all moveable objects move one step to their facing direction.
3) UP: all controlled objects move one step up, and all moveable objects move one step to their facing direction.
4) DOWN: all controlled objects move one step down, and all moveable objects move one step to their facing direction.
5) SPACE: all moveable objects move one step to their facing direction, and **NO** controlled objects move.

E. Winning and Dead End

The winning condition is any player contacts any winning object, and the player still survives after contacting. Specifically, both conditions need to be satisfied at the same time:

1) 'X' in 'X-IS-YOU' contacts 'Y' in 'Y-IS-WIN'.
2) The number of 'X' must be greater than zero after contacting.

Since players must move to push words and interact with the environment, the game will enter a dead end if players cannot control anything on the map, and the player must restart the level in this case. To be more specific, the game will enter a deaf end if at least one of the following is satisfied:

1) No rule of format 'X-IS-YOU' exists.
2) The number of 'X' in 'X-IS-YOU' becomes zero.

IV. Baselines for *Baba Is Y'all*

Because of the timeliness and specificity of *Baba Is You*, there is almost no present algorithm discussing methods and ideas about its solution. The only well-performed algorithm is the best-first tree search solver in the game module of *Baba Is Y'all* framework, along with traditional search algorithms: BFS, DFS, and default agent - provided by organizers of "Keke AI Competition".

A. Default Agent

This algorithm is a best-first tree search algorithm and is first developed by *Baba Is Y'all* framework, which is used to evaluate the quality of a user-made level.

The algorithm uses a heuristic function based on the Manhattan distance to the centroid distance for three groups: keyword objects, objects associated with the 'WIN' rule, and objects associated with the 'PUSH' rule. These three categories are chosen because of their critical importance in solving process: winning objects are required to complete the level, keyword objects are responsible for rules manipulation, and pushable objects can directly or indirectly affect the layout of a level map and accessibility to reach winning objects.

The following equation represents the heuristic function:

$$h = \frac{n + w + p}{3}$$

The meaning of each variable is shown in Table.III.

The first baseline, the default agent in Keke AI Competition, originates from this algorithm. It is almost the same as the above best-first tree search. The only difference is that the default agent takes the average of all possible Manhattan distances in each category instead of the minimal nearest Manhattan distance.

The difference can be seen in Table.IV.

The pseudocode of the primary process is presented in Algorithm 1.

TABLE III: Variables in *Baba Is Y'all Framework*

Variable	Definition
h	The final heuristic value
n	The minimum Manhattan distance from any player object to the nearest winnable object
w	The minimum Manhattan distance from any player object to the nearest word block
p	The minimum Manhattan distance from any player object to the nearest pushable object

TABLE IV: Variables of default agent in *Keke AI Competition*

Variable	Definition
h	The final heuristic value
n	The average Manhattan distance from any player object to any winnable object
w	The average Manhattan distance from any player object to any word block
p	The average Manhattan distance from any player object to any pushable object

B. BFS / DFS Agent

As two of the most classic graph search algorithms, once the game board is represented as a graph, it can be explored using either BFS (breadth-first search) or DFS (depth-first search) and find a solution. BFS is typically better suited for finding the shortest path to a solution, while DFS is better for finding any solution with generally more calculation resources.

However, it is worth noting that the search space for "Baba is You" can be quite large, especially for more complex levels, so using heuristics or other optimizations may be necessary to find a solution in a reasonable amount of time.

Algorithm 1 Default Agent

Input: $init\_state$ - initial game state;
Output: $solution$ - action list $[a_0, ..., a_n]$ to win the game;

```
function ITERSOLVE(init_state)
    heap.push(newNode(init_state))
    while heap is not empty do
        curNode ← heap.pop()
        [score, children] ← getChildren(curNode)
        for child in children do
            if child win then
                return child.actions
            end if
            if child died then
                children.remove(child)
            end if
        end for
        heap.pushAll(children)
        sort heap in descending order by score
    end while
end function
```

C. Random Agent

The algorithm will randomly choose one action from five possible legal actions, repeat the generation 50 times, and return an action list with a length of 50.

V. Agents based on Search Algorithm

Besides four baselines, we proposed two other agents based on two different search algorithms: Rolling Horizon Evolutionary Algorithms(RHEA) and Monte Carlo Tree Search(MCTS).

A. RHEA

RHEA is a subclass of Evolutionary Algorithm which evolve action sequences for games; it has the advantage of finding locally optimal solutions quickly based on the current state considering only a short period of future steps, which makes it ideal for single-player real-time games [8]. It has also been proved that RHEA performed well in general video game playing [9] [10].

The RHEA algorithm begins the game by initializing a population of P individuals, each representing a randomly generated action sequence of length L. These individuals are evaluated by simulating the action sequence, which means starting from the current game state and performing actions in sequence in order. The final game state reached is evaluated using a heuristic function, and the state value becomes the individual's fitness.

For each generation, the best E individuals(lowest fitness) are promoted directly to the next generation. The rest of the $P - E$ individuals are offspring generated by following procedures: for each offspring, K individuals will be chosen randomly from the previous generation, then the best two of them will become parents, and crossover is applied to create one offspring, which will mutate and be added into next generation. Then the next generation will replace the previous generation, and all individuals are evaluated as described above heuristic function, then this generation process repeats.

After reaching the appointed number of iterations or limitations of time, RHEA returns the first action in the best individual as the selected action to play in the game. For any subsequent game ticks, the previous final population won't be carried through to the next game tick.

The pseudocode of the RHEA process is described in Algorithm 2.

B. MCTS

Monte Carlo Tree Search(MCTS) is a heuristic algorithm with significant performance in many decision games. This algorithm does not require expert knowledge or specific problem rules of the game. Instead, it relies on simulation and statistical methods for decision-making. In general, the MCTS algorithm will start from the current game state, then choose a simulation direction with the highest discovery value and do plenty of simulations in this direction, then choose the action with the highest winning rate during simulation.

More specifically, the process of MCTS can be divided into four parts: selection, expansion, simulation, and backpropagation.

Algorithm 2 RHEA Agent

Input: $init\_state$ - initial game state;
Output: $solution$ - action list $[a_0, ..., a_n]$ to win the game;

```
function ITERSOLVE(init_state)
    cur_node ← newNode(init_state)
    while time and iterations within limitations do
        action ← ACT(cur_node)
        actions.push(action)
        cur_node ← nextNode(cur_node, action)
        if win then
            return actions
        end if
    end while
    return actions
end function

function ACT(cur_state)
    pop ← init_population(P)
    evaluate(pop, cur_state)
    while time and iterations within limitations do
        next_pop ← getBest(pop, E)
        for i from 1 to P − K do
            candidates ← randomPick(pop, P − E)
            parents ← getBest(candidates, 2)
            new_individual ← crossover(parents)
            new_individual ← mutate(new_individual)
            next_pop.push(new_individual)
        end for
        pop ← new_pop
    end while
    return getBestAction(pop)
end function
```

Selection is the core idea of MCTS. The selection starts from the root node containing current game state information and checks its five children based on five legal actions. The purpose of this step is to find the child who is least discovered so that the algorithm can search uniformly like BFS; that is, the child that hasn't been expanded has higher priority than children that have already been expanded. So, on the one hand, if a child hasn't been explored, it can be chosen directly. Here we randomly choose an action among all children that haven't explored to import a random element. On the other hand, if all five children have been expanded, then the following evaluation function, Upper Confidence Bound Apply to Tree(UCT), will be implemented for all children, and the one with the maximal value will be chosen. The meaning of each variable is explained in Table V. Note that the UCT value will be greater if a child node is less explored than others.

$$UCT(i) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

The expansion is performed only when there's no child in the selected direction. In this case, the algorithm will perform the selected action based on the current node and store moved information in the child node.

TABLE V: Variables in UCT function

Variable	Definition
w_i	The number of winning times of child.
n_i	The number of simulation times of the child.
N_i	The number of simulation times of the current node.
c	A weight number.

The simulation is also called rollout or playout. Starting from the expanded(or selected) node, many random actions are chosen and performed on the node continuously until it reaches the winning or dead end. Then a heuristic function is implemented on the final state.

The simulation result will backpropagate along all of its parents. For each parent, the simulation times and the fitness value will be updated according to the heuristic value of the final state. Until then, an overall MCTS iteration is done.

The iteration of these four processes will repeat many times until reaching the ending condition or finding a correct answer. The pseudocode of while MCTS process is described in Algorithm 3.

Algorithm 3 MCTS Agent

Input: $init\_state$ - initial game state;
Output: $solution$ - action list $[a_0, ..., a_n]$ to win the game;

```
function ITERSOLVE(init_state)
    cur_state ← init_state
    while time and iterations within limitations do
        action ← ACT(cur_state)
        actions.push(action)
        cur_state ← nextState(cur_state, action)
        if win then
            return actions
        end if
    end whilereturn actions
end function

function ACT(cur_node)
    while time and iterations within limitations do
        action ← selection(cur_node)
        new_node ← expansion(cur_node, action)
        fitness ← simulation(new_node)
        backpropagation(newNode, fitness)
    end while
    return getBestAction(cur_node)
end function
```

VI. EXPERIMENTS

A. Algorithms

Seven algorithms are tested in the experiment, including four baselines(default, BFS, DFS, random), two search algorithms (RHEA, MCTS), and the winner algorithm of Keke AI Competition in 2022 [11].

All algorithms using heuristic function adopt the same function as described in Section IV-A, that is, $h = \frac{n+w+p}{3}$,

where n is the average Manhattan distance from any player object to any winnable object, w is average to any word block, and p is average to any pushable object.

Particularly, for the MCTS algorithm, a multiple-beginning strategy is used. In the result below, 'MCTS-10' refers to the fact that this MCTS version has ten beginnings, and each is separated from others, 'MCTS-1' refers to the original version, which has only one beginning to search.

B. Dataset

Although Keke AI Competition offers some demo levels for testing [1], they are too small as a dataset for experiments.

As referred before, *Baba Is Y'all* is an online level editor website allowing players to upload their own created levels. This website also offers download functionality to obtain all levels in JSON format. This dataset is approximately 250 levels, enough for the experiment.

C. Result

Referring to Keke AI Competition, all algorithms have the constraint of 10,000 iterations and 10.0s. The experiment results are shown in Table.VI. The meanings of indices are as follows:

1) Agents: the algorithm for the agent.
2) Accuracy: the percentage of successfully solved levels to total levels.
3) Avg. runtime (s): the average time spent per level, including unresolved levels, in seconds.
4) Avg. solution length: the average length of the solution for each level, including unsolved levels.
5) Rank: rank under the rules of Keke AI Competition in 2022

TABLE VI: Experimental Result

Agents	Accuracy	Avg. runtime (s)	Avg. solution length	Rank
MCTS-1	79.30%	2.464	99.8	1
MCTS-10	78.50%	2.450	94.8	2
2022 Best	65.40%	4.028	93.2	3
Random	59.80%	1.176	50.0	4
Default	57.70%	4.484	15.3	5
BFS	56.10%	4.518	25.9	6
DFS	48.2%	5.506	332.4	7
RHEA	29.3%	4.689	8.6	8

D. Analysis

1) Baselines: All four baselines show correct properties. Random does not include the search process, so it has a minimal runtime for each level, and its solution length is all 50.0. DFS regards depth as the highest priority, so it has the maximal solution length among all algorithms. In comparison, BFS regards breadth as the highest priority, so its solution length is much less than DFS. The default agent is the only algorithm with a heuristic function among four baselines, so it has the highest accuracy.

2) MCTS: MCTS reaches the highest accuracy among all algorithms. The multiple advantage properties of MCTS can explain this.

Compared to traditional search algorithms, MCTS is more efficient. It evaluates the value of each action by simulating random games rather than searching the entire game tree. This scalability and computational efficiency make MCTS particularly effective in handling complex problems like "Baba Is You".

Another reason is that MCTS has a stronger exploration ability. It employs random simulation to explore different decision paths extensively. This allows it to perform well in complex, unknown, or uncertain environments and discover better solutions than the four baselines.

3) Multiple-beginning MCTS: In the above result, the MCTS algorithm using and not using multiple beginnings have almost the same accuracy. After many other experiments, it can be concluded that there is no obvious difference between these two types, and the tiny difference in accuracy can be regarded as an error within the acceptable range.

One possible explanation is that even for MCTS with multiple starting points, each still uses the same algorithm. The advantage of multiple starting points over single starting points is that they can be simulated more times at the same depth and are more likely to achieve the optimal value at the same depth. For the rules of the game Baba Is You, the single-beginning MCTS is already very close to the optimal value, so multiple-beginning is no longer an advantage.

4) RHEA and MCTS: It is observed that the results of RHEA and MCTS differ greatly for the same search type of algorithm. One possible explanation is that RHEA, which applies to general video games, does not apply to puzzle games.

RHEA considers only short-time sequences of future actions, which makes it less considerate, more efficient to solve, and very real-time.RHEA also bases its decisions on the current environment's changing state, allowing it to explore solutions that apply to the current state. Both of these factors make it suitable for use in real-time fighting games.

However, puzzle games emphasize solving results rather than immediacy, so RHEA's greatest advantage no longer exists.

In the iterative process, RHEA introduces new individuals randomly and then detects whether the unique individuals apply to the current state. There is almost no connection between the two individuals. Despite crossover and mutation, the parents of crossover are still randomly generated, so the relationship between the two individuals is minimal.

MCTS, on the other hand, first determines all possible actions in the current state and then allocates the same search resources to each step based on these actions as much as possible. This search type is more systematic, comprehensive, and uniform. Combined with the 10-second and 10,000-iteration limits, RHEA does not explore enough random solutions in a limited time, so MCTS is far more accurate than RHEA.

VII. Conclusion and Future Improvement

The demand for intelligent agents arises as the puzzle's complexity keeps increasing. Although many algorithms perform well in traditional puzzles with static rules, dynamic rule puzzles remain to be explored. Future tests on one of the latest games, , will conduct based on the programming environment provided by Keke AI Competition.

In the first stage, we clarified the game's rules, finished configuring the game simulation environment, and successfully ran several of the given baselines to prove it.

In the second phase, we implemented two search category algorithms, RHEA and MCTS, and tested them in an expanded test set. Among them, RHEA performed poorly because it did not apply to this game, while MCTS outperformed any other algorithm, including the competition's winner of last year.

In the future, in the third phase, we plan to try to solve the game levels using reinforcement learning. Since the current common packages are based on Python, we will encounter problems such as cross-language calls. If possible, we will configure the reinforcement learning environment in phase 3 and try to use some algorithms to achieve better results.

References

[1] J. T. M Charity, Sarah Chen, "Keke AI competition," http://keke-ai-competition.com/, 2022.

[2] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*. Springer, 2018, vol. 2.

[3] R. Canaan, H. Shen, R. Torrado, J. Togelius, A. Nealen, and S. Menzel, "Evolving agents for the hanabi 2018 cig competition," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.

[4] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*. IEEE, 2013, pp. 320–323.

[5] M. Charity, A. Khalifa, and J. Togelius, "Baba is y'all," http://equius.gil.engineering.nyu.edu/, 2022.

[6] S. Bozyel, "Keke AI competition," https://github.com/sonelb/Keke-AI/blob/main/agents/random_BABA_GO_solvesRiverone_AGENT.js, 2022.

[7] M. Charity, A. Khalifa, and J. Togelius, "Baba is y'all: Collaborative mixed-initiative level design," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 542–549.

[8] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 351–358. [Online]. Available: https://doi.org/10.1145/2463372.2463413

[9] R. D. Gaina, D. Perez-Liebana, S. M. Lucas, C. F. Sironi, and M. H. Winands, "Self-adaptive rolling horizon evolutionary algorithms for general video game playing," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 367–374.

[10] R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez-Liebana, "Rolling horizon evolutionary algorithms for general video game playing," *IEEE Transactions on Games*, vol. 14, no. 2, pp. 232–242, 2022.

[11] J. T. M Charity, Sarah Chen, "Keke AI competition," https://github.com/MasterMilkX/KekeCompetition, 2022.