Computers and Education: Artificial Intelligence 2 (2021) 100014

Contents lists available at ScienceDirect

Computers and Education: Artificial Intelligence

journal homepage: www.elsevier.com/locate/caeai

ColorShapeLinks: A board game AI competition for educators and students

Nuno Fachada *

Lusófona University, COPELABS / HEI-Lab, Campo Grande, 376, Lisbon, Portugal

ARTICLE INFO

Keywords:
Board games
Videogame development
Computer games
Simplexity
AI competition
AI agent
AI curriculum
DotNet

ABSTRACT

ColorShapeLinks is an AI board game competition framework specially designed for students and educators in videogame development, with openness and accessibility in mind. The competition is based on an arbitrarily-sized version of the Simplexity board game, the motto of which, "simple to learn, complex to master", is curiously also applicable to AI agents. ColorShapeLinks offers graphical and text-based frontends and a completely open and documented development framework built using industry standard tools and following software engineering best practices. ColorShapeLinks is not only a competition, but both a game and a framework which educators and students can extend and use to host their own competitions. It has been successfully used for running internal competitions in AI classes, as well as for hosting an international AI competition at the IEEE Conference on Games.

1. Introduction

ColorShapeLinks is an artificial intelligence (AI) competition framework for the Simplexity board game (Brain Bender Games, 2009) with arbitrary dimensions. It is a similar game to Connect-4, with pieces defined not only by color, but also by shape.

The ColorShapeLinks development framework offers Unity (Unity Technologies, 2021) and .NET console frontends. Agents are implemented in C# and run unmodified in either frontend. Unity and C# are widely used in the games industry (Toftedahl & Engström, 2019) and for game development education (Comber et al., 2019; Dickson, 2015; Dickson et al., 2017; Fachada & Códices, 2020; Hmeljak, 2020), making the competition especially accessible to this audience. The framework is open source, fully documented and developed following best practices in software engineering, allowing it to be studied and extended by educators, researchers and students alike (Jiménez et al., 2017; Lakhan & Jhunjhunwala, 2008). Furthermore, it contains all the tooling for setting up competitions. It can be used for internal competitions in AI courses, for example, or for running international AI competitions, as demonstrated in the 2020 edition of the IEEE Conference on Games (IEEE CoG). In this regard, ColorShapeLinks is not simply a software tool that uses AI technologies for educational purposes (Chen et al., 2020), but a software toolkit to both teach and learn board game AI.

This paper is organized as follows. In Section 2, the state of the art in board game AI—and how ColorShapeLinks fits in—is briefly discussed. The ColorShapeLinks board game and its original version, Simplexity, are characterized in Section 3. In Section 4, the motivation for developing ColorShapeLinks, as well as the educational context in which the framework took form, are examined. The development framework, namely its architecture, existing frontends, agent implementation, included agents, and availability, are reviewed in Section 5. In Section 6, we describe how ColorShapeLinks was used to host two internal competitions in an AI course unit, as well as a fully-fledged international AI competition in the IEEE CoG 2020 conference. The implications and limitations of the framework and of the reported outcomes are discussed in Section 7. The paper closes with Section 8, in which we present some conclusions.

2. Background

Computer programs for playing classical board games such as Chess, Draughts or Go, were the first known application of AI for games (Millington, 2019). AI research in games has subsequently grown far beyond the domain of board games (Yannakakis & Togelius, 2018). Nonetheless, these continue to be a focus of active research, not only in AI techniques for playing particular games—e.g., Go (Silver et al., 2016)—but also for playing board games in general (Konen, 2019; Kowalski et al., 2019; Piette et al., 2020), or even creating new ones (Kowalski & Szykuła, 2016; Stephenson et al., 2019). Furthermore, new board game AI challenges and competitions continue to be proposed (Justesen et al., 2019).

While board games are probably one of the easiest ways to introduce AI for games to students (Chesani et al., 2017; Drake & Sung, 2011),

* Corresponding author:
E-mail address: nuno.fachada@ulusofona.pt.

https://doi.org/10.1016/j.caeai.2021.100014
Received 22 November 2020; Received in revised form 16 February 2021; Accepted 17 February 2021

state-of-the-art board game AI research is gaining some distance from both industry and education in videogame development. Requirements such as general game playing capabilities for the AI or knowledge of general game specification languages (Kowalski et al., 2019; Stephenson et al., 2019) can potentially raise the entry level for newcomers and/or discourage prospective participants which could otherwise bring new ideas to academia—or at least get involved in its processes.

ColorShapeLinks aims to reduce this gap by offering an approachable, open and flexible AI competition for the videogame development education audience. This is accomplished by making the development of an AI agent very simple (e.g., write a single method and test it in Unity), while allowing for considerable customization via a modular framework architecture. This same modularity enables games and full tournaments to run within and without Unity. While accessible for undergraduates and non-specialist game developers, ColorShapeLinks provides a challenging and intricate board game competition, addressable with vastly different techniques, from knowledge-based methods (Allis, 1988) to machine learning techniques (Silver et al., 2016), or anything in between.

It should be noted that ColorShapeLinks could be implemented under a general game playing system such as RBG (Kowalski et al., 2019) or Ludii (Piette et al., 2020), and this would probably be an excellent exercise. However, using these frameworks would mean losing some of the educational advantages offered by ColorShapeLinks, namely accessibility, openness (e.g., Ludii was not open source at the time of writing) and keeping the focus on general game design and development education while providing students with a broad AI for games background.

3. ColorShapeLinks

ColorShapeLinks is a version of the Simplexity board game with arbitrary and parameterizable dimensions. We describe Simplexity and ColorShapeLinks in Subsections 3.1 and 3.2, respectively, highlighting their computational characteristics in Subsection 3.3.

3.1. The Simplexity board game

The Simplexity board game is similar to Connect-4 in two regards: 1) the first player to connect four pieces of a given type wins; and, 2) the base board size is 6×7. The crucial difference is that, in Simplexity, pieces are defined not only by color, but also by shape—round or square. Player 1, *white*, wins if it can connect either four round pieces or four white pieces. Likewise for player 2, *red*, but with square or red pieces. Players begin with 21 pieces of their color, 11 of which square, and the remaining 10 round. The catch here is that shape has priority over color as a winning condition. Therefore, a player can lose in its turn when placing a piece of the opponent's winning shape. Table 1 summarizes these rules and Fig. 1 shows the possible winning conditions.

3.2. The ColorShapeLinks board game

ColorShapeLinks is a Simplexity game parameterizable with respect to board dimensions, number of pieces required for a winning sequence, and initial number of round and square pieces. Table 2 shows the available parameters as well as the symbols used for them in the remainder of this paper.

Table 1
Simplexity rules.

	Player 1	Player 2
Plays first?	Yes	No
Plays with	White pieces	Red pieces
Begins with	11× white square pieces	11× red square pieces
	10× white round pieces	10× red round pieces
Wins with line of	4× round pieces	4× square pieces
(shape has priority)	4× white pieces	4× red pieces

3.3. Characteristics

ColorShapeLinks can be characterized as follows (Millington, 2019; Yannakakis & Togelius, 2018):

- It is a **turn-based** game.
- It is a **two-player**, **zero-sum adversarial** game.
- It is **deterministic**, i.e., there is no random element influencing the game state.
- It has **perfect information**, i.e., the board state is fully observable.
- The **maximum number of turns** is given by

  $$t_{\max} = \min\{r \cdot c, 2(s+o)\}$$

  i.e., it is equal to the minimum between the number of board positions, $r \cdot c$, and the total pieces available to be played, $2(s+o)$.

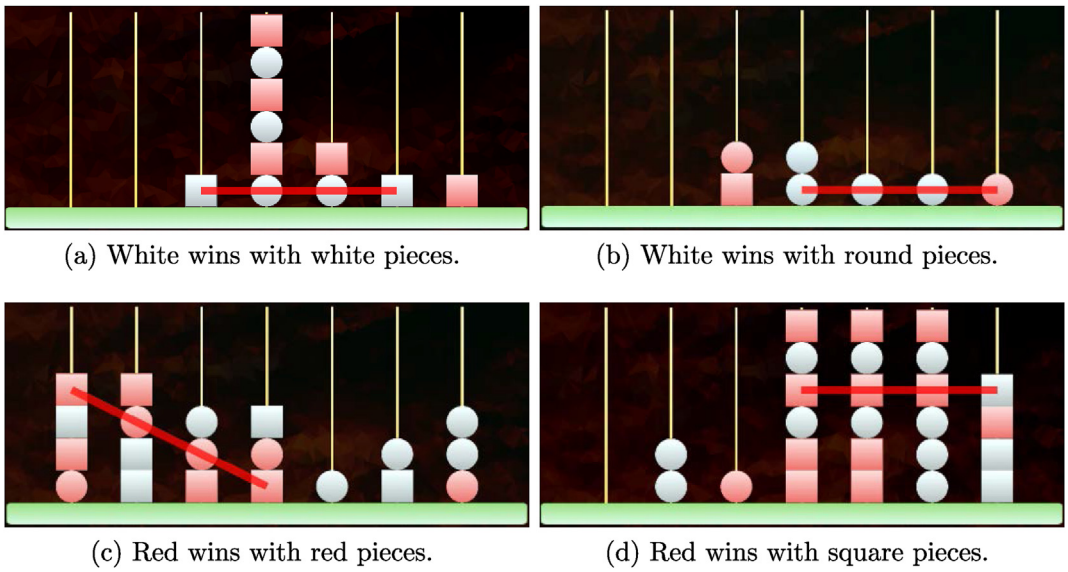
- Since each board position can be in one of five states (empty, white circle, white square, red circle or red square), an upper bound for the **state space** is given by $5^{t_{\max}}$. In practice the state space will be considerably smaller, since this value includes invalid states, for example when pieces are on top of empty cells.
- The initial **branching factor** is $2 \times c$ (2 shapes, c columns), although it may decrease during the game as columns are filled and pieces of a certain shape are played-out. Consequently, an upper bound for the game tree size is given by $(2 \cdot c)^{t_{\max}}$.

These characteristics place ColorShapeLinks in an interesting position for AI research. The game, even with standard Simplexity parameters, cannot currently be solved using a brute force approach in a short amount of time. Like most games nowadays, machine learning techniques (e.g., deep learning) together with a tree search approach such as Monte Carlo Tree Search (MCTS) (Coulom, 2007), for example, will certainly be able to produce hard-to-beat agents. However, the limited and well-defined ruleset leaves the door open for knowledge-based or even analytical solutions.

4. Educational context and motivation

ColorShapeLinks was originally developed as a Unity-only assignment for an AI for Games course unit[1] at Lusófona University's Videogames BA. This is an evenly interdisciplinary degree (Mateas & Whitehead, 2007), meaning that while it possesses solid Computer Science (CS) fundamentals, it is more limited in this regard than technology-focused curriculums, giving equal ground to Game Design and Art courses. The Unity game engine is used in most of the course units, since it is easy to learn, free, cross-platform, and widely used in education and actual game development (Comber et al., 2019; Dickson, 2015; Dickson et al., 2017; Fachada & Códices, 2020; Hmeljak, 2020; Toftedahl & Engström, 2019). Consequently, and to increase student's proficiency with Unity, the engine's scripting language, C#, is lectured independently in two programming course units (Fachada & Códices, 2020). The AI for Games course unit threads this fine line by implementing a hands-on, Unity and industry-oriented program based on selected parts of Millington's *AI for Games* textbook (Millington, 2019). The course unit addresses topics such as heuristics, board games, movement, pathfinding, decision making, learning and procedural content generation. However, more complex and/or academic materials are avoided. With respect to the topic of board games, the course focuses on the Minimax algorithm, as well as several variations and optimizations, such as alpha-beta pruning, move ordering and iterative deepening.

[1] https://github.com/VideojogosLusofona/ia_2019_board_game_ai.



(a) White wins with white pieces. (b) White wins with round pieces.

(c) Red wins with red pieces. (d) Red wins with square pieces.

Fig. 1. Possible victory conditions in ColorShapeLinks using standard Simplexity rules.

Table 2
ColorShapeLinks parameters.

Symbol	Name	Default[a]
r	Rows	6
c	Columns	7
w	Win sequence	4
s	Square pieces per player	11
o	Round pieces per player	10

[a] i.e., a regular game of Simplexity.

Techniques such as transposition tables and MCTS are not currently lectured, although in the specific case of MCTS this might change given the proven usefulness of the method in a variety of different games and scenarios (Yannakakis & Togelius, 2018).

Due to its wide-ranging design, Lusófona's Videogames degree attracts students from various areas of study and with substantially different interests. An issue with this type of curriculum design is that it can be difficult to motivate students enrolled in courses outside their main preferences. This is often the case of art and design-inclined students in CS courses in general (Fachada, 2018, 2020; de Andrade & Fachada, 2020), and AI in particular (Lin et al., 2021). Thus, student motivation strategies become particularly relevant in this context (Pintrich, 2003). Board game competitions in AI courses have been shown to motivate students in studying course materials and in autonomously search for solutions beyond the scope of the course's program and lectures (Chesani et al., 2017). Hence, and with the goal of increasing motivation among students—particularly those whose primary interests do not lie in CS—a board game AI assignment featuring an internal classroom competition was devised. Given the variety of student backgrounds, the project had to be made accessible for everyone, while challenging for more advanced and/or CS-oriented students. A deterministic, fully observable two-player, non-trivial board game was an obvious choice. Another requirement was that it was not a very well-known game, so that not much AI code was available online, forcing students to be original. Simplexity (Brain Bender Games, 2009) ticks those boxes, as it was only used once in an AI course (Wilkins, 2012), to our knowledge. Furthermore, Simplexity was implemented as a console C# project (no AI) two years prior,[2] thus being a perfect choice, since students already knew the base game. This led to the development of ColorShapeLinks, an AI assignment and competition, first introduced in the first semester of the 2019/20 academic year.

Moreover, some authors have reported positive educational outcomes when creating assignments based on international AI competitions (Kim & Cho, 2013; Yoon & Kim, 2015). They argue that, by offering students the possibility of submitting their solutions to international events, and of comparing their solutions with the state of the art, their engagement and motivation are improved. Following this line of reasoning, ColorShapeLinks was proposed as a competition for the IEEE CoG 2020 conference. Since the first version of the framework was only suitable for basic classroom competitions, it was extended to support more advanced use cases. Additions included a console mode, for advanced debugging and analysis of agents, and a set of scripts for setting up automatically running tournaments. Thus, ColorShapeLinks recast its role as an internal competition during the second semester of the 2019/20 academic year, while at the same time hosting the eponymous competition at the IEEE CoG 2020 conference. These experiences are described in Section 6.

Beyond the educational benefits of ColorShapeLinks discussed thus far, and the fact that Unity is widely used in education and industry, it should also be noted that: (1) C# is used as a scripting language in several major game engines (Unity Technologies, 2021; Bello et al., 2021; Crytek, 2021; Linietsky et al., 2021; Spilman et al., 2021; FlaxEngine, 2021; UNIGENE Corp., 2021; Plain Concepts, 2021); and, (2) neither Unity nor C# are very common in game AI competitions. Thus, ColorShapeLinks has the potential to more generally reduce the gap between academic AI and game development industry/education. This is especially relevant for interdisciplinary and/or industry-oriented programs such as Lusófona's Videogames BA.

5. Development framework

The ColorShapeLinks development framework offers two application frontends, one for the Unity game engine, the other a text-based affair aimed at terminal use. This allows students familiar with Unity to start implementing an agent from their comfort zone, advancing to the text-based console frontend if Unity development limitations begin to show. Advanced students or researchers can skip the Unity frontend altogether, and start with the console frontend from the offset. The framework architecture supporting this flexibility is described in Subsection 5.1. Subsection 5.2 details the two available frontends, highlighting their advantages and disadvantages. The basics of implementing an agent are discussed in Subsection 5.3, while the agents included with the

[2] https://github.com/VideojogosLusofona/lp1_2017_p1.

framework are presented in Subsection 5.4. Finally, the framework's availability, documentation and licensing are outlined in Subsection 5.5.

5.1. Architecture

The development framework is organized around three main components, namely the Unity frontend (UnityApp, a single .NET/Mono project), the console frontend (ConsoleApp, composed of two .NET projects), and the Common .NET project. This organization is shown in Fig. 2, and discussed with additional detail in the following paragraphs.

5.1.1. The Common project

The Common project is a .NET class library which constitutes the core of the framework. It defines the fundamental models—from a Model-View-Controller (MVC) perspective (Sarcar, 2020)—of the ColorShapeLinks game, such as the board, its pieces or performed moves, and is a dependency of the remaining projects. It is further subdivided in the AI and Session namespaces. The former defines AI-related abstractions, such as the `AbstractThinker` class, which AI agents must extend, as well as a manager for finding and instantiating concrete AI agents. The latter specifies a number of match and session-related interfaces, as well as concrete match and session (i.e., tournament) models.

5.1.2. The ConsoleApp frontend projects

The ConsoleApp is composed of two .NET projects, App and Lib, both of which depend on the Common class library, as shown in Fig. 2. The App project is a .NET console application with an internal dependency on the Lib project, itself a .NET class library. The App project provides the actual console frontend, namely the text user interface (TUI) with which the user interacts in order to run ColorShapeLinks matches and sessions.

The Lib class library acts as an UI-independent "game engine", offering match and session controllers, as well as interfaces for the associated event system, allowing to plug in renderers (views, in MVC parlance) or other event handling code at runtime. It serves as a middleware between the Common library and frontend applications, such as the one implemented in the App project. It is not used by the Unity implementation, since Unity already provides its own game engine logic, forcing match and session controllers to be tightly integrated with its frame update cycle. Nonetheless, the Lib class library makes the creation of new ColorShapeLinks TUIs or GUIs very simple, as long as they are not based on highly prescriptive frameworks such as Unity.

5.1.3. The UnityApp frontend project

The UnityApp is a ColorShapeLinks frontend implemented in the Unity game engine. Like the ConsoleApp, it is designed around the MVC design pattern, making use of the models provided by the Common library. In this case, however, the views and controllers are tightly integrated with the Unity engine.

5.2. Frontends

The two available frontends, for Unity and console, have similar capabilities. Both are capable of performing matches and tournaments involving multiple AI agents and human players. An agent can be used without modification when moving from one frontend to the other. There are, however, advantages in using the console frontend. The following paragraphs offer additional detail on each frontend.

5.2.1. The Unity frontend

In the Unity frontend, matches and tournaments are played within the Unity Editor, though it is possible to create a standalone build with prefixed match or tournament configurations. The rationale behind this choice is that ColorShapeLinks is at its core a development framework. Therefore, it makes sense this is done within the Unity Editor, which is a development environment. A game of ColorShapeLinks running within the Unity Editor is shown in Fig. 3.

Developing an agent within the Unity editor has the disadvantage that games run considerably slower than in the console. Furthermore, constantly creating standalone builds with prefixed configurations is not practical. Therefore, while this frontend makes ColorShapeLinks approachable, it should be considered more of an introduction to the competition than a definitive way of implementing advanced state-of-the-art ColorShapeLinks agents.

5.2.2. The console frontend

The console UI allows for a more refined control of ColorShapeLinks matches and sessions. Being based on MVC, it allows for easily swapping the UI, as well as running matches at full speed, contrary to the Unity editor. Fig. 4 shows running a ColorShapeLinks match using the console UI.

While the console frontend offers many extensibility points, its default configuration will likely suffice in most situations. For example, users can run a learning algorithm on top of the console app, making use of its return values, which indicate the match result or if an error occurred.

5.3. Implementing an agent

The first step to implement an AI agent is to extend the `AbstractThinker` base class. This class has three overridable methods, but it is only mandatory to override one of them, as shown in Table 3.

There is also the non-overridable `OnThinkingInfo()` method, which can be invoked for producing "thinking" information, mainly for debugging purposes. In the Unity frontend this information is printed on Unity's console, while in the console frontend the information is forwarded to the registered thinker listeners (or views, from a MVC perspective).

Classes extending `AbstractThinker` also inherit a number of useful read-only properties, namely board and match configuration properties (number of rows, number of columns, number of pieces in sequence to win a game, number of initial round pieces per player and number of initial square pieces per player) and the time limit for the AI to play. Concerning the board/match configuration properties, these are

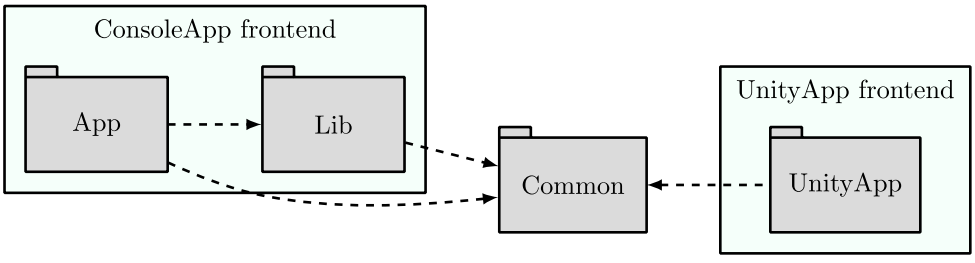


Fig. 2. Internal organization of the ColorShapeLinks development framework. Arrows represent dependencies between separate .NET projects.

Fig. 3. Running ColorShapeLinks using the Unity Editor UI.

```
$ dotnet run -- match -W ColorShapeLinks.Common.AI.Examples.MinimaxAIThinker \
                      -R ColorShapeLinks.Common.AI.Examples.RandomAIThinker

=> MinimaxD3 (White) vs Random (Red) <=

        w - White Round
        W - White Square
        r - Red   Round
        R - Red   Square

MinimaxD3 (White) turn
MinimaxD3 (White) placed a Round piece at column 3 after 186ms
.......
.......
.......
.......
.......
...w...

Random (Red) turn
Random (Red) placed a Square piece at column 5 after 0ms
.......
.......
.......
.......
.......
...w.R.
```

Fig. 4. Running ColorShapeLinks using the console UI. Only the start of the match is shown.

Table 3
Overridable Methods in the `AbstractThinker` class.

Method	Mandatory override?	Purpose
`Setup()`	No	Setup the AI agent.
`Think()`	Yes	Select the next move to perform.
`ToString()`	No	Return the AI agent's name.

also available in the board object given as a parameter to the `Think()` method. However, the `Setup()` method can only access them via the inherited properties.

The following subsections address the overriding of each of these three methods.

5.3.1. *Overriding the* `Setup()` *method*

If an AI agent needs to be configured before starting to play, the `Setup()` method is the place to do it. This method receives a single argument, a string, which can contain agent-specific parameters, such as maximum search depth, heuristic to use, and so on. It is the agent's responsibility to parse this string. In the Unity frontend, the string is specified in the "Thinker params" field of the AIPlayer component. When using the console frontend, the string is passed via the `-white/red-params` option for simple matches, or after the agent's fully qualified name in the configuration file of a complete session. Besides the parameters string, the `Setup()` method also has access to board/match properties inherited from the base class.

The same AI agent can represent both players in a single match, as well as more than one player in sessions/tournaments. Additionally, separate instances of the same AI agent can be configured with different parameters. In such a case it might be useful to also override the `ToString()` method for discriminating between the instances configured differently. This is an essential feature if ColorShapeLinks is running under a machine learning and/or optimization infrastructure.

Note that concrete AI agents require a parameterless constructor in

order to be found by the various frontends. Such constructor exists by default in C# classes if no other constructors are defined. However, it is not advisable to use a parameterless constructor to setup an AI agent, since the various board/match properties will not be initialized at that time. This is yet another good reason to perform all agent configuration tasks in the `Setup()` method. In any case, concrete AI agents do not need to provide an implementation of this method if they are not parameterizable or if they do not require an initial configuration step.

5.3.2. *Overriding the* `Think()` *method*

The `Think()` method is where the AI actually does its job and is the only mandatory override when extending the `AbstractThinker` class. This method accepts the game board and a cancellation token, returning a `FutureMove` object. In other words, the `Think()` method accepts the game board, the AI decides the best move to perform, and returns it. The selected move will eventually be executed by the match engine.

The `Think()` method is called in a separate thread. Therefore, it should only access local instance data. The main thread may ask the AI to stop thinking, for example if the thinking time limit has expired. Thus, while thinking, the AI should frequently test if a cancellation request was made to the cancellation token. If so, it should return immediately with no move performed.

The game board can be freely modified within the `Think()` method, since this is a copy and not the original game board being used in the main thread. More specifically, the agent can try moves with the `DoMove()` method, and cancel them with the `UndoMove()` method. The board keeps track of the move history, so the agent can perform any sequence of moves, and roll them back afterwards. For parallel implementations, the agent can create additional copies of the board, one per thread, so that threads can search independently of each other.

The `CheckWinner()` method of the game board is useful to determine if there is a winner. If there is one, the solution is placed in the method's optional parameter. For building heuristics, the game board's public read-only variable `winCorridors` will probably be useful. This variable is a collection containing all corridors (sequences of positions) where promising or winning piece sequences may exist.

The AI agent will lose the match in the following situations:

- Causes or throws an exception.
- Takes too long to play.
- Returns an invalid move.

5.3.3. *Overriding the* `ToString()` *method*

By default, the `ToString()` method removes the namespace from the agent's fully qualified name, as well as the "thinker", "aithinker" or "thinkerai" suffixes. However, this method can be overridden in order to behave differently. One such case is when agents are parameterizable, and differentiating between specific parameterizations during matches and sessions becomes important.

5.3.4. *Summing-up*

Building an AI agent for ColorShapeLinks is very simple, asking only the implementer to extend a class and implement one method. A basic Minimax algorithm with a simple heuristic can be implemented in less than 30 minutes. Educators can use it to demonstrate how to create a simple agent from scratch, during a class for example, and leave up to the students to find better, more efficient solutions. This can be done as an assignment, a competition, or both.

5.4. *Included agents*

Three test agents are included with the framework, serving both as an example on how to implement an agent, as well as a baseline reference for testing other agents.

The *sequential* agent always plays in sequence, from the first to the last column and going back to the beginning, although skipping full columns. It will start by using pieces with its winning shape, and when these are over, it continues by playing pieces with the losing shape. Therefore, it is not a "real" AI agent.

The *random* agent plays random valid moves, avoiding full columns and unavailable shapes. It can be parameterized with a seed to perform the same sequence of moves in subsequent matches (as long as the same valid moves are available from match to match).

The *minimax* agent uses a basic, unoptimized Minimax algorithm with a naive heuristic which privileges center board positions. It can be parameterized with a search depth, and, although simple, is able to surprise unseasoned human players—even at low search depths.

5.5. *Availability*

The development framework is available at https://github.com/VideojogosLusofona/color-shape-links-ai-competition and is fully open source, licensed under the Mozilla Public License 2.0[3] (MPL2), which requires changes to the source code to be shared, although allowing for integration with proprietary code if the MPL2 licensed code is kept in separate files. The framework is completely documented and the documentation is available at https://videojogoslusofona.github.io/color-shape-links-ai-competition/docs/html/.

6. Deployments

The ColorShapeLinks framework has been used to host two internal competitions (in separate semesters) in an AI for Games course unit at Lusófona University's Videogames BA, as well as an international AI competition in the IEEE CoG 2020 conference. These deployments are discussed in the next two subsections.

6.1. *Internal competitions in AI for games course units*

ColorShapeLinks was used as an assignment and internal competition in an AI for Games course unit during the two semesters of the 2019/20 academic year. In both semesters, the submitted AI agents and associated reports were graded preliminarily and separately from the final competition. Since students work in groups of 2 or 3 elements, an individual discussion was also performed. Results from the internal competition were used to potentially improve the preliminary grades, not lower them, since the main goal was to motivate students and have an engaging class where students watched and commented on the performance of each others' agents in real time. Nonetheless, if an agent was not competent enough to enter the competition, i.e., it freezed the Unity project or did not respond to cancellation requests, up to 1 point could be subtracted from the final grade. No students from the first semester repeated the course during the second semester.

However, there were several differences between the two semesters. In the first semester, ColorShapeLinks was a Unity-only assignment, as stated in Section 4. The minimum requirement for passing—i.e., to have a grade of 10 or higher (grades are given in a 0–20 scale)—was that students implemented an agent capable of defeating the *sequential* and *random* agents. The basic *minimax* agent, described in Subsection 5.4, was not included in the framework at this time. In the second semester, the console frontend and the basic *minimax* agent were added to the framework, with the assignment coinciding with the first few weeks of the international competition. The minimum passing requirement in the second semester was for the submitted agent to beat the basic *minimax* implementation. Grade distribution and summary statistics for the ColorShapeLinks assignment in both semesters are shown in Fig. 5 and Table 4, respectively.

Looking at the overall results in both semesters, at least 80% of the students had a passing grade (≥ 10) and participated actively in the

[3] https://www.mozilla.org/en-US/MPL/2.0/.

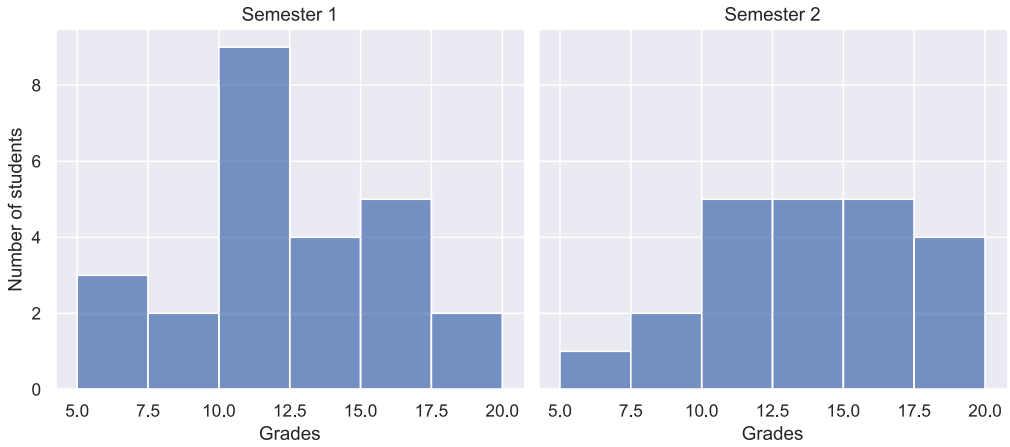


Fig. 5. Grade distribution for the ColorShapeLinks AI assignment in the two semesters of the 2019/20 academic year. Grades are given in a 0–20 scale.

Table 4
Summary statistics for the ColorShapeLinks AI assignment grades in the two semesters of the 2019/20 academic year, namely, number of students enrolled, n_{all}, number of students evaluated in the assignment, n_{eval}, mean grade, $\bar{x}$, median grade, $\tilde{x}$, and percentage of students with a passing grade, $\%_{\geq 10}$. Grades are given in a 0–20 scale.

	n_{all}	n_{eval}	$\bar{x}$	$\tilde{x}$	$\%_{\geq 10}$
Semester 1	27	25	12.02	12.00	80.00%
Semester 2	25	22	13.64	13.00	86.36%

assignment. Students less interested in the AI aspect of game design and development mostly submitted Minimax-based agents, e.g. with alpha-beta pruning, just to pass the project. They were, nonetheless, generally more committed than in other AI and programming assignments. On the other end, students more comfortable with programming and AI generally dedicated more hours to the assignment and came up with interesting and competitive agents. Most, if not all students were highly engaged with the final live competition during class (e.g., "my AI is better than yours"!).

There were some notable grading variations between the two semesters. Grades in the second semester were higher in average, and more students met the passing grade (Table 4), even though the minimum requirement was higher. Grade distribution (Fig. 5) shows that, in the first semester, a considerable number of students aimed at the bare minimum just to get a passing grade, while in the second semester students generally went for more elaborate solutions. Some of these went beyond the scope of the lectured materials, making use of transposition tables, search parallelization and heuristics parameter learning. We argue there were three main reasons, which separately or in combination, led to this outcome:

1. The console app allowed some students to better test and debug their agent implementations, without being constrained by Unity.
2. Some students might have been motivated due to working on an assignment which was at the same time an international AI competition.
3. The *minimax* agent allowed students to study and better understand how a "real" agent could be implemented, allowing them to build upon it.

These results suggest that ColorShapeLinks was successful in allowing students with various interests and with different capabilities within game making—as is the case of students in Lusófona's Videogames BA—to be able to produce palpable, working AI code, and to feel included when addressing this challenging topic.

6.2. IEEE CoG 2020 international competition

ColorShapeLinks was accepted as an official AI competition at the IEEE CoG 2020 conference, and was funded with a prize money of 500 USD for the winner of each track. The competition ran on two distinct tracks:

1. The *Base Track*, which used standard Simplexity rules with a time limit of 0.2 seconds per move. Only one processor core was available for the AI agents.
2. The *Unknown Track*, which was played on a multi-core processor under a parameterization known only after the competition deadline, since it was dependent on the result of a public lottery draw.

The goal of the *Base Track* was to test agent capabilities in the standard Simplexity game. The *Unknown Track* evaluated the generalization power of the submitted solutions when applied to a most likely untested parameterization.

For each track, the submitted agents played against each other two times, so each had the opportunity of playing first. Agents were awarded 3 points per win, 1 point per draw and 0 points per loss, with the final standings for each track depending on the total number of points obtained per agent.

Classifications for the *Base Track* were updated daily during a five month period, up until the submission deadline, together with two larger test parameterizations. This allowed participants to have an idea of how their submission was faring, and update it accordingly.

The competition had a total of six submissions, four of which were from undergraduate students, both solo and in teams. Two of the submissions were from students of the author which fared well in the internal competition discussed in the previous subsection. Although the number of submissions was low, the fact that four of them were from undergrads, partially demonstrated that the competition was accessible.

Eita Aoki, from Japan, won the *Base Track* with his *Thunder* agent, which used MCTS together with a custom bit board implementation. The winner and runner-up of the *Unknown Track* were teams from Portugal and students of the author which attended the AI for Games course in the second semester. João Moreira won the track with *SureAI*, a highly optimized Minimax-based agent. The *SimpAI* agent, developed by a team of three students, was the runner-up, implementing a set of hand-crafted heuristics balanced with an evolutionary algorithm (Fernandes et al., 2021). While interesting, none of these agents was truly state-of-the-art, leaving the door open for better agents going forward.

7. Implications and limitations

The deployments discussed in the previous sections demonstrate the potential and usefulness of the ColorShapeLinks framework. The internal competition appeared to motivate the students, who were highly engaged during the class tournaments held in both semesters. The introduction of the international competition during the second semester had a more pronounced effect, though. Overall grades and dedication improved, with students generally showing more autonomy and going beyond what was lectured in the classes, effectively validating the works of Kim and Cho (2013) and Yoon and Kim (2015). The authors of *SimpAI*, a student group of the second semester, were able to publish a paper describing their solution and the second place it obtained in the *Unknown Track* of the international competition (Fernandes et al., 2021). These results show that AI competitions in general, and ColorShapeLinks in particular, have clear educational benefits with respect to student motivation, engagement and autonomy. This is further highlighted by the fact that Lusófona's Videogames degree is interdisciplinary (non-CS focused) and industry-driven (not academy-oriented). The fact that students from such a degree were able to compete in an international competition with good results indicates that ColorShapeLinks shows good promise in bridging the gap between game development education and academic AI.

However, the research presented in this paper has two main limitations which should be highlighted. First, no survey was done in order to assess students' reflections on the competitions. The stated student motivation and engagement during the internal competitions are taken from the author's observations, and thus, entirely subjective and potentially biased, no matter how clear or obvious they might have been. Second, the underlying game has a very narrow focus. Even as an unbounded version of Simplexity, ColorShapeLinks—as a game—is relatively simple. Thus, many good and excellent solutions are bound to appear on the short to medium term, possibly even analytical solutions, such as the case of Connect-4 (Allis, 1988). Consequently, ColorShapeLinks—as a framework—might have a short service life before the problem it addresses becomes effectively solved. Still, this is not the case at the time of writing, as ColorShapeLinks will again be one of the competitions being held at the IEEE CoG 2021 conference.

8. Conclusions

In this paper we presented ColorShapeLinks, and AI board game competition framework specifically designed for game development students and educators, with openness and accessibility at its core. The arbitrarily-sized Simplexity board game—implemented by the framework—offers a good balance between simplicity and complexity, being approachable by undergraduates, while posing a non-trivial challenge to researchers and more advanced students. The framework has been successfully used for running internal competitions in an AI for Games course unit, as well as for hosting an international AI competition, validating its usefulness. Although the problem addressed by the proposed framework is bound to be solved in the next few years—thus rendering the framework obsolete—the general ideas presented in this paper, such as running internal and international AI competitions using industry standard tools and software best practices for educational purposes, were confirmed, and remain valid for similar future endeavors.

Statements on open data and ethics

The data reported in this paper is fully anonymized and publicly available at the Zenodo open-access scientific repository (Fachada, 2021).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by Fundação para a Ciência e a Tecnologia under Grants UIDB/04111/2020 (COPELABS) and UIDB/05380/2020 (HEI-Lab). The author would like to thank André Fachada for proofreading the text. The author would also like to thank the anonymous referees for their valuable comments and helpful suggestions.

References

Allis, L. V. (1988). *A knowledge-based approach of connect-four. The Game is solved: White wins*. Netherlands: Master's thesis Vrije Universiteit Amsterdam.

de Andrade, D., & Fachada, N. (2020). Fun maths for all game development students. In *Proceedings of the 2020 ACM conference on Innovation and technology in computer Science education ITiCSE '20* (pp. 529–530). New York, NY, USA: ACM. https://doi.org/10.1145/3341525.3393992.

Bello, V., et al. (2021). *Stride*. URL: https://stride3d.net/.

Brain Bender Games. (2009). Simplexity. Discovery bay games. https://boardgamegeek.com/boardgame/55810/simplexity.

Chen, X., Xie, H., & Hwang, G.-J. (2020). A multi-perspective study on artificial intelligence in education: Grants, conferences, journals, software tools, institutions, and researchers. *Computers & Education: Artificial Intelligence, 1*, Article 100005. https://doi.org/10.1016/j.caeai.2020.100005. URL: http://www.sciencedirect.com/science/article/pii/S2666920X20300059.

Chesani, F., Galassi, A., Mello, P., & Trisolini, G. (2017). A game-based competition as instrument for teaching artificial intelligence. In F. Esposito, R. Basili, S. Ferilli, & F. A. Lisi (Eds.), *AI*IA 2017 Advances in artificial intelligence lecture Notes in computer Science* (pp. 72–84). Springer International Publishing. https://doi.org/10.1007/978-3-319-70169-1_6.

Comber, O., Motschnig, R., Mayer, H., & Haselberger, D. (2019). Engaging students in computer science education through game development with unity. In *2019 IEEE Global engineering education conference (EDUCON) (pp. 199–205). IEEE*. https://doi.org/10.1109/EDUCON.2019.8725135

Concepts, P. (2021). Wave engine. URL: https://waveengine.net/.

Coulom, R. (2007). Efficient selectivity and backup operators in monte-carlo tree search. In H. J. van den Herik, P. Ciancarini, & H. H. L. M. J. Donkers (Eds.), *Computers and Games* (pp. 72–83). Berlin, Heidelberg: Springer Berlin Heidelberg volume 4630 of Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-540-75538-8_7. international Conference on Computers and Games. CG 2006.

Crytek. (2021). CryEngine. URL: https://www.cryengine.com/.

Dickson, P. E. (2015). Using unity to teach game development: When you've never written a game. In *Proceedings of the 2015 ACM conference on Innovation and technology in computer Science education ITiCSE '15* (pp. 75–80). New York, NY, USA: ACM. https://doi.org/10.1145/2729094.2742591.

Dickson, P. E., Block, J. E., Echevarria, G. N., & Keenan, K. C. (2017). An experience-based comparison of unity and unreal for a stand-alone 3d game development course. In *Proceedings of the 2017 ACM conference on Innovation and technology in computer Science education ITiCSE '17* (pp. 70–75). New York, NY, USA: ACM. https://doi.org/10.1145/3059009.3059013.

Drake, P., & Sung, K. (2011). Teaching introductory programming with popular board games. In *Proceedings of the 42nd ACM Technical Symposium on computer Science education SIGCSE '11* (pp. 619–624). New York, NY, USA: ACM. https://doi.org/10.1145/1953163.1953338.

Fachada, N. (2018). Teaching database concepts to video game design and development students. *Revista Lusófona de Educação, 40*, 151–165. https://doi.org/10.24140/issn.1645-7250.rle40.10. URL: http://revistas.ulusofona.pt/index.php/rleducacao/article/view/6437.

Fachada, N. (2020). Desafios no ensino da programação a alunos de videojogos. In *Atas do 1.° SEVj - Seminário Sobre Ensino de Videojogos* (pp. 59–73). Sociedade Portuguesa de Ciências dos Videojogos. URL: http://hdl.handle.net/10316/89402.

Fachada, N. (2021). *ColorShapeLinks 2019/20 grades dataset*. Zenodo. https://doi.org/10.5281/zenodo.4543679 (Version 1.0.0) [Data set].

Fachada, N., & Códices, N. (2020). Top-down design of a CS curriculum for a computer games BA. In *Proceedings of the 2020 ACM conference on Innovation and technology in computer Science education ITiCSE '20* (pp. 300–306). New York, NY, USA: ACM. https://doi.org/10.1145/3341525.3387378.

Fernandes, P. A., Inácio, P. A., Feliciano, H., & Fachada, N. (2021). SimpAI: Evolutionary heuristics for the ColorShapeLinks board game competition. In I. Barbedo, J. P. Sousa, & B. Legerén (Eds.), *Videogame Sciences and arts, VJ 2020 Communications in computer and information Science*. Springer (in press).

FlaxEngine. (2021). Flax engine. URL: https://flaxengine.com/.

Hmeljak, M. (2020). Developing a computer graphics course with a game development engine. In *Proceedings of the 2020 ACM conference on Innovation and technology in computer Science education ITiCSE '20* (pp. 75–81). New York, NY, USA: ACM. https://doi.org/10.1145/3341525.3387428.

Jiménez, R. C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., Capella-Gutierrez, S., Hong, N. C., Cook, M., Corpas, M., et al. (2017). Four simple recommendations to encourage best practices in research software [version 1; peer review: 3 approved]. *F1000Research*, 6. https://doi.org/10.12688/f1000research.11407.1

Justesen, N., Uth, L. M., Jakobsen, C., Moore, P. D., Togelius, J., & Risi, S. (2019). Blood bowl: A new board game challenge and competition for ai. In *2019 IEEE conference on Games (CoG)* (pp. 1–8). IEEE. https://doi.org/10.1109/CIG.2019.8848063.

Kim, K.-J., & Cho, S.-B. (2013). Game ai competitions: An open platform for computational intelligence education [educational forum]. *IEEE Computational Intelligence Magazine, 8*, 64–68. https://doi.org/10.1109/MCI.2013.2264568

Konen, W. (2019). General board game playing for education and research in generic AI game learning. In *2019 IEEE conference on Games (CoG) (pp. 1–8). IEEE.* https://doi.org/10.1109/CIG.2019.8848070

Kowalski, J., Mika, M., Sutowicz, J., & Szykuła, M. (2019). Regular boardgames. In , *Vol. 33. Proceedings of the Thirty-Third AAAI conference on artificial intelligence* (pp. 1699–1706). of AAAI-19.

Kowalski, J., & Szykuła, M. (2016). Evolving chess-like games using relative algorithm performance profiles. In G. Squillero, & P. Burelli (Eds.), *Applications of evolutionary computation* (pp. 574–589). Springer International Publishing volume 9597 of Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-319-31204-0_37. european Conference on the Applications of Evolutionary Computation).

Lakhan, S. E., & Jhunjhunwala, K. (2008). Open source software in education. *Educause Quarterly, 31*, 32–40. URL: https://er.educause.edu/articles/2008/5/open-source-software-in-education.

Lin, P.-Y., Chai, C.-S., Jong, M. S.-Y., Dai, Y., Guo, Y., & Qin, J. (2021). Modeling the structural relationship among primary students' motivation to learn artificial intelligence. *Computers & Education: Artificial Intelligence, 2*, Article 100006. https://doi.org/10.1016/j.caeai.2020.100006. URL: https://www.sciencedirect.com/science/article/pii/S2666920X20300060.

Linietsky, J., Manzur, A., et al. (2021). Godot engine. URL: https://godotengine.org/.

Mateas, M., & Whitehead, J. (2007). Design issues for undergraduate game-oriented degrees. In *Proceedings of the 2007 Microsoft academic Days on Game development in computer Science education* (pp. 85–89).

Millington, I. (2019). In *AI for Games* (3rd ed.). Boca Raton, FL, USA: CRC Press https://doi.org/10.1201/9781351053303.

Piette, E., Soemers, D. J., Stephenson, M., Sironi, C. F., Winands, M. H., & Browne, C. (2020). Ludii – the ludemic general game system. In G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, & J. Lang (Eds.), *ECAI 2020* (pp. 411–418). volume 325 of Frontiers in Artificial Intelligence and Applications. https://doi.org/10.3233/FAIA200120.

Pintrich, P. (2003). A motivational science perspective on the role of student motivation in learning and teaching contexts. *Journal of Educational Psychology, 95*, 667–686. https://doi.org/10.1037/0022-0663.95.4.667

Sarcar, V. (2020). In *Design patterns in C#: A hands-on Guide with real-world examples* (2nd ed.). Apress. https://doi.org/10.1007/978-1-4842-6062-3.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*, 484–489. https://doi.org/10.1038/nature16961

Spilman, T., et al. (2021). *MonoGame*. URL: http://www.monogame.net/.

Stephenson, M., Piette, E., Soemers, D. J., & Browne, C. (2019). Ludii as a competition platform. In *2019 IEEE conference on Games (CoG)* (pp. 1–8). IEEE. https://doi.org/10.1109/CIG.2019.8848084.

Toftedahl, M., & Engström, H. (2019). A taxonomy of game engines and the tools that drive the industry. In A. Nakamura (Ed.), *DiGRA '19 - Proceedings of the 2019 DiGRA international conference: Game, play and the Emerging Ludo-Mix*. DiGRA: Digital Games Research Association.

UNIGENE Corp. (2021). Unigene. URL: https://unigine.com/.

Unity Technologies. (2021). Unity®. URL: https://unity.com/.

Wilkins, D. (2012). *Program 4a: Simplexity. CSCI 531: Artificial intelligence*. Computer and Information Science Department, University of Mississippi. https://john.cs.olemiss.edu/~dwilkins/CSCI531/fall12/prog4.html.

Yannakakis, G. N., & Togelius, J. (2018). *Artificial intelligence and Games*. Springer. http://gameaibook.org.

Yoon, D.-M., & Kim, K.-J. (2015). Challenges and opportunities in game artificial intelligence education using angry birds. *IEEE Access, 3*, 793–804. https://doi.org/10.1109/ACCESS.2015.2442680