Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 192 (2021) 1914–1923

Procedia Computer Science

www.elsevier.com/locate/procedia

25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

The application of selected modern artificial intelligence techniques in an exemplary strategy game

Rafał Dreżewski[a,*], Jakub Solawa[a]

[a]*AGH University of Science and Technology, Institute of Computer Science, Cracow, Poland*

Abstract

In the paper requirements for artificial intelligence algorithms designed for modern strategy computer games are analyzed. The selected techniques used to fulfill those requirements are described in detail. Then the exemplary game, which was designed and implemented using the selected algorithms for path-finding, decision-making, tactical and strategic reasoning, and team coordination is presented. Finally, the results of experiments conducted with the use of the created game are analyzed. The results prove the efficiency of selected techniques in creating a strategically challenging game.

Keywords: artificial intelligence; computer games; strategy games; team coordination; decision-making

1. Introduction

For generations humans are occupied with different games. Games are generally associated with fun and innocent frivolity, and usage of this word may give an impression of them being a trivial matter. However, it could be argued that many important issues, like war, elections or personal relationships are games. Moreover, because of their nature, games are an interesting application area from the point of view of artificial intelligence—it is easy to define straightforward rules, the game's state is simple to represent and in most cases the number of available moves is significantly restricted. Games are a good candidate for a problem that can be optimally solved by artificial intelligence algorithms and therefore can be successfully used to test and verify many artificial intelligence techniques.

However, artificial intelligence techniques used in game development usually have a different goal than to find the best solution. Instead of maximizing the chances of success, or optimally solving the problem, they are used to create a convincing experience for the player. Computer games are very diverse and "convincing experience" means something very different for each of them, thus in each of them the AI will function differently. The methods used in constructing AI are developed based on a very broad variety of sources, heavily modifying them. Additionally, game

[*] Corresponding author.
E-mail address: drezew@agh.edu.pl

developers introduce their alterations in many ways, and therefore the techniques used to solve similar problems can drastically differ with every creator.

This paper aims to show the researched methods used in the process of developing artificial intelligence in computer games. Because of the overwhelming variety of those techniques, it has been decided to reduce the field of research to strategy games. Presented methods are going to be tested in the experiment—a simple strategy computer game with an implemented AI module. The developed game will be verified experimentally for the quality of the AI module and its ability to be a worthy opponent for human players with different experience level.

2. A review of selected artificial intelligence methods for strategy computer games

There are several essential artificial intelligence requirements for real-time strategy games [8]: path-finding, decision-making, tactical and strategic reasoning, and group movement.

Those requirements are also very similar for turn-based strategy games. The most significant difference is in the movement—there is no need for kinematic steering, as units can be transported directly [8].

2.1. Path-finding

Characters in strategy games usually move around a level map. Because they hardly ever know in advance where they will have to move, it is impossible to implement them a fixed route. A unit in a real-time strategy game may be ordered by the player to move to any point on the map at any time. For each of those characters the game AI module must be capable of finding an appropriate route through the game world to allow the character to traverse the level map and reach the target [6].

This capability is called path-finding. In most cases, it is used to figure out where to move to reach a target. The target itself is decided by another part of the game AI module—the decision-making algorithm. The path-finder's only task is to establish the route leading to the goal [12].

The majority of games relies on path-finding solutions based on an algorithm called A* [8, 1]. A* is efficient and not hard to implement, however, it is incapable of working directly with the game level data. The game level is required to be represented in a form of a directed non-negative weighted graph.

2.2. Decision-making

Decision-making is all about the capability of a character to decide on its next action. The character processes a set of information, which is then used to generate an action that needs to be performed. The input to the decision-making system is the knowledge that a character possesses, and the output is an action request. The knowledge can be organized into external and internal knowledge [8].

External knowledge is the information that a figure possesses about the surrounding environment. The examples of such data could be the position of other characters, the layout of the level, or the direction that a sound is coming from. Internal knowledge is information about the character's internal state or thought processes, for example its health, its goals, or previously executed actions.

Likewise, actions can consist of two components. An action requested can change the external state of the character (for example, attacking another figure or changing its position) or it can only impact the character's internal state. Changes to the internal state are not as apparent in game applications but are still important in some decision-making algorithms. They might include adjusting the character's opinion of the player, updating its emotional state, or changing its current goal.

Decision trees are one of the simplest and most straightforward decision-making techniques. They are usually not extremely sophisticated, but because of their flexibility they have many uses in decision-making. The decision tree is used to obtain an action from a set of actions. This action must correspond to the input—the situation when the decision is made. The tree groups numerous inputs together under one action, while the inputs keep the capability to influence the output—the decision [9].

Usually, characters in a game will perform actions from a limited set. They keep doing the same thing until an event or external influence forces them to change their behavior. For example, AI-controlled sentinel will stand in its post until it notices the player, then it will enter the combat mode, taking cover and shooting the player. **State machines** are the technique often used for this kind of decision-making [8]. They make up the vast majority of decision-making systems used in games nowadays. State machines take into consideration both the world around them and their internal state [2].

2.3. *Tactical and strategic reasoning*

In the decision-making techniques presented above, the decisions are made only by a single character, using only its knowledge. The character does not utilize the information about the whole strategic situation of its team. The same techniques can be used for making decisions in a wider, strategic spectrum, but first this strategic information must be obtained and interpreted in some way.

The waypoint is a position in the game, which a character can occupy [5]. Waypoints correspond to the vertices in the path-finding methods described earlier in this section. To allow a waypoint to be useful outside path-finding and movement, it is essential for it to contain additional data of tactical importance. When a character occupies the waypoint with tactical features it can use those features to its advantage [8].

Strategy computer games almost always use multiple characters working together to accomplish their goals. Players, both human an AI-controlled, lead the armies consisting of a significant number of units. To be effective and become a worthy challenge for a human player, those units need to synergize their actions.

The most straightforward approach to coordinated action is to create an AI on multiple levels. For example, in such a multi-leveled AI every unit has its AI, a team of those units has a separate AI, and a division containing many teams also has an independent AI [11].

There is more than one way for such a multi-tier AI to operate. In the *top-down* approach, the AI on the highest tier makes its decision, and then informs the next tier about it. This lower tier then utilizes this information to make its own decision, notifies the next tier, and so on until the lowest tier. In the *bottom-up* approach AI of the lowest tier makes the autonomous decisions, consulting the AI from the higher tiers to obtain the information used for choosing its actions [8].

2.4. *Group movement*

Another requirement for AI in strategy computer games is to move characters in the game in a sensible fashion [7]. This requirement applies mainly to real-time strategy games. Turn-based strategy games often do not need movement techniques—when a decision is made where to move, the character can simply be placed in the target location.

The most commonly adopted technique of steering characters in strategy computer games are kinematic movement algorithms [8]. Kinematic movement algorithms use static data (position and orientation, without velocities) and produce a requested velocity. The output is often simply a target direction, and one of two velocities: full speed of the character or remaining stationary. Kinematic algorithms do not use acceleration.

In strategy computer games groups of characters are often required to move in a coordinated manner. Coordination of motion can happen at two levels. The individual figures can make decisions that complement each other, resulting in their movements being organized. The decision can also be made as a whole team to move in an established and coordinated group.

The movement of many figures in a way that they maintain a certain level of group cohesion is called formation motion [8]. There are a few different approaches to introduce formation motion to characters in a game. This gives this technique a degree of flexibility, while remaining simple and efficient in producing sensible behavior.

The easiest method of implementing formation motion is to use fixed geometric formations. This kind of formation consists of a set of slots, which are locations where a character can be positioned [8].

Another approach to formation motion is to give every character its movement mechanism and then allow them to calculate their position based on the position of other characters [8].

Emergent formations have the benefit of enabling each character to react individually to obstacles and collisions. There is no need to consider the size of the formation while turning or avoiding obstacles because all the figures in the formation will follow the rules. The disadvantage of this approach is the difficultly to set up the rules to acquire a desired shape. The result is usually a little chaotic, not as strict as in the fixed formation.

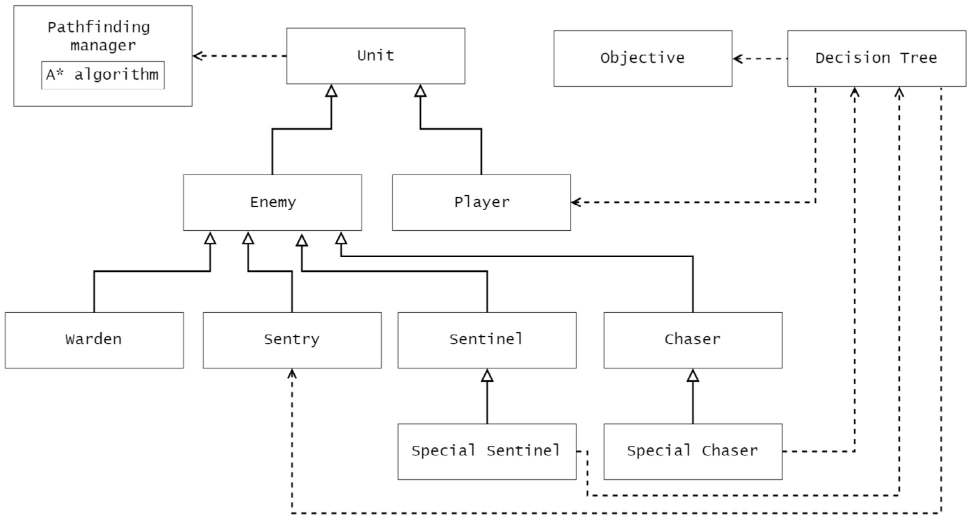

Fig. 1. Simplified model of the game's architecture.

3. An exemplary strategy game with AI mechanisms

To test how effective the presented AI methods are in creating an engaging experience they were used in an experimental strategy game.

3.1. Game summary

The game was titled *Blitz: Tactical Action*. It is a real-time strategy game, where the player takes control of up to three units. The main goal of the game is to collect the objectives scattered on the game's level. The objectives are guarded by AI-controlled units, and player characters are unable to obtain the objectives until there are no guards near the objective. The opposing units can shoot each other, dealing some damage with each shot that hits. When a unit, both player or AI-controlled, takes enough damage it is destroyed and removed from the game. When all the objectives are picked up, the player wins the game. If all the player's units are destroyed, he loses the skirmish. All the units, either player or AI-controlled, along with objectives are represented by colored squares.

The game was created with the help of *Unity* game engine. All of the game's code, including the artificial intelligence code, was written in *C#* language.

3.2. Game architecture

Each non-player unit is controlled by its own state machine. The units are divided into several types, each with different behavior—different state machine (see Fig. 1). The whole team of AI-controlled units is supervised by a higher-tier AI—a decision tree. The decision tree monitors the course of the game, and can give orders to Special Sentinel and Special Chaser-type units to respond to different situations. The tree requires information from objectives, player units and Sentry-type units to make its decisions.

Both player and AI-controlled units can request a path to be calculated by a path-finding manager. The manager implements the A* algorithm using the Euclidean distance as an estimating heuristic. The found path is returned to a unit, which then uses kinematic movement to follow the path.

3.2.1. Strategic decision tree

Every frame the instance of a decision tree checks for notifications about situations that might require a response. These notifications are sent to the tree by player characters (player was destroyed), objectives (objective is endangered or was stolen) or its units (Sentry-type unit detected a player character)—see Fig. 2.

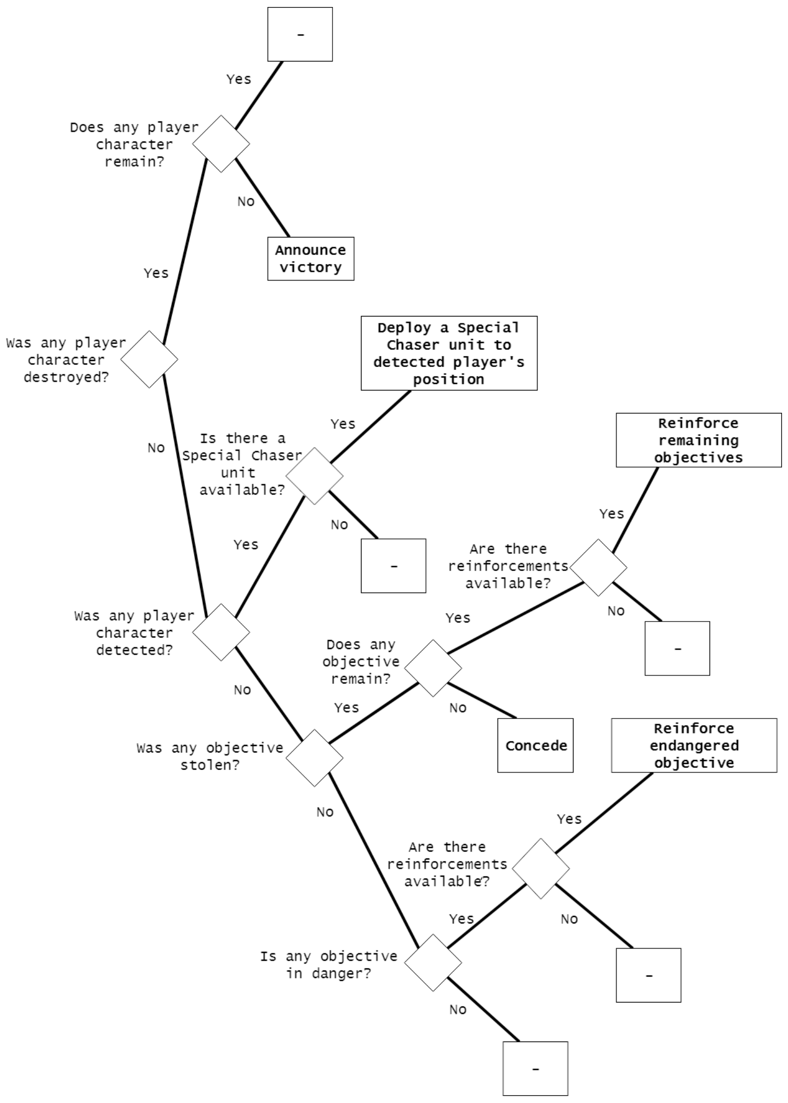
-
Yes
Does any player character remain?
No
Announce victory
Yes
Was any player character destroyed?
Deploy a Special Chaser unit to detected player's position
Yes
No
Is there a Special Chaser unit available?
Reinforce remaining objectives
Yes
No
-
Yes
Are there reinforcements available?
Was any player character detected?
Yes
No
-
Does any objective remain?
No
Yes
No
Concede
Was any objective stolen?
Reinforce endangered objective
No
Yes
Are there reinforcements available?
Yes
No
Is any objective in danger?
-
No
-

Fig. 2. Strategic decision tree.

When a player character takes lethal damage and is therefore destroyed, it notifies the tree. The tree then checks if the destroyed player character was the last one, and if yes, it informs the player, that the game is finished and the AI is victorious.

Upon detecting a player character, Sentry-type unit alerts the tree and gives it the detected player's location. If the tree has an available Special Chaser unit, it sends this unit to the obtained location.

Objective warns the decision tree when it is stolen. The tree then counts the remaining objectives and if there are none left, it ends the game and congratulates the player on victory. However, if at least one objective is still in play, the AI orders available Special Sentinel-type units to relocate to those objectives.

The target can also signal the tree when there are no guards nearby, making it vulnerable to being captured by the player. In such a situation AI sends a Special Sentinel-type unit to reinforce the objective, if only a unit is available.

3.2.2. State machine-controlled characters

There are six types of AI-controlled characters in the game: Warden, Sentinel, Sentry, Chaser, Special Sentinel, and Special Chaser.

The Warden is the simplest unit. It is stationary and does not move around the map. Warden constantly scans the surrounding area, and if a player character enters this area it starts shooting this character. The Warden will stop shooting when all the player units left the area or were destroyed. The squares representing Warden-type units are purple.

The levels in the game contain obstacles, and waypoints neighboring with those obstacles have the tactical characteristic of providing cover from at least one of four cardinal directions. The Sentinel is one of two unit types in the game (the other one being a Special Sentinel), which can utilize this information.

The Sentinel patrols the area, which means it successively moves between two predefined locations. If it finds the player character nearby, it determines from which direction this character is approaching and then moves to a waypoint providing appropriate cover. Once the Sentinel is in cover, it starts to shoot the player. Once there are no player units in the vicinity, Sentinel resumes patrolling. Sentinels are pictured by orange squares.

The Sentry is the only unit controlled by a state machine, which can pass the information to the strategic decision tree. It patrols the area just like the Sentinel-type units. The moment it detects the player-controlled unit, the Sentry informs the decision tree about the location of this detected unit. It is enough to pass this information to the tree once for it to make its decision, therefore, to avoid flooding it with redundant data—then the Sentry waits for 5 seconds. During this time it is unable to move or to detect player characters. Once the 5 seconds pass, the Sentry proceeds with its patrol, and can detect and alarm again. Sentries are illustrated by yellow squares, and during the 5 seconds of waiting after alarming the strategic AI, are highlighted with red.

Chasers patrol their surroundings the way Sentinels and Sentries are. When Chaser finds a player unit during its patrol it starts moving towards it and shooting. The Chasers usually move slower than player characters, so it is possible to move out of the Chaser's detection range. If the Chaser does not find any player unit nearby, it returns to its patrol. The color of squares used to represent Chaser-type units is red.

Special Sentinels are the first unit to be controlled by a hierarchical state machine. When strategic decision-making mechanism decides that an objective requires help, it orders the Special Sentinel to move to the objective. The unit obeys the order no matter what state it was previously in. When the relocation is finished, the unit resumes its behavior, which is identical to those of the Sentinel-type unit. Special Sentinels are portrayed as brown squares.

Along with Special Sentinels, Special Chasers use a hierarchical state machine to make decisions. If the Sentry-type unit detects the player and gives his location to the decision tree, the tree orders a Special Chaser-type unit to move to this location. When this Special Chaser fulfills this command, it starts to act just like a Chaser unit. Special Chaser-type units are dark red.

3.2.3. Pathfinding Manager

Units send requests to find a path between two locations, one of them being usually the unit's current position. The path-finding manager adds every request to a queue, and processes them one at a time. Once the path is found, it is returned to the requesting unit.

The manager uses the A* algorithm with Euclidean distance heuristic to fulfill those requests. The use of this heuristic means that the estimated cost of the edge between vertices is calculated as the distance between the representative point of each vertex, even if there are no direct connections. This heuristic can be precise or underestimating [8]. Avoiding walls and other obstacles only adds extra distance. If there are none, then the heuristic is precise.

3.3. Scoring system

Whenever the game is over, both the player and the AI are awarded points. The maximum score for both the player and AI is 300 points. The same rules are applied no matter which side wins. The winner is awarded 60 points. For each objective obtained by the player, he is awarded 50 points. Moreover, whenever the player takes an objective, 15 points are added for each surviving unit besides the first. For example, if the player takes the first objective while controlling three units, the second one with two units and the last one with only one unit, he will be awarded 255 points—150 for all three objectives, 60 for winning the game, 30 for taking the first objective with three units remaining and 15 for taking the second one with two units remaining.

For each objective which the player did not manage to obtain, the AI is awarded 50 points. Furthermore, the percentage of remaining AI-controlled units is multiplied by 90 points, and rounded down if it is not an integer. The score is then increased by the resulting value. For example, if the player is defeated after obtaining one objective and

6 out of 9 starting AI-controlled units remained, the AI would get 220 points—100 for two remaining objectives, 60 for winning the game and 60 for surviving units.

3.4. Gameplay

To conduct the experiment, three different levels in the game were designed. When the game is launched, the main menu is shown, and it is possible to start a level chosen by a player.

The first level of the game is at the same time the game's tutorial. The controls and rules of the game are shown on the obstacles. Player units are represented by numbered blue squares. Objectives are pictured by green squares, and when guarded, highlighted with yellow. There is also a few enemies. The player is supposed to familiarize himself with the controls and then defeat the enemies and collect the objectives, finishing the level.

To demonstrate that the objective is impossible to obtain, until there are no enemies near it, one objective is protected by a single Warden-type unit which is easy to defeat, and in contrast one objective is completely unguarded. The level also features one Sentinel-type enemy to show that certain enemies can hide behind obstacles. To present an enemy with a different behavior and to provide some challenge for the learning player, two Chasers patrol the final objective. There are four enemies in total, which means that each surviving AI-controlled unit is worth 22.5 points for the AI.

In the second level the player starts with his units separated from each other and placed in different corners of the level. This enables the player to consider many approaches to beat the game level. The player can try to collect the objectives, each guarded by a single Warden, while keeping his units separated, without the advantage in the number of units. The player can also start by trying to unite all his units, but this approach will risk Sentries alarming the Special Chaser units waiting at the bottom of the level. This level also features Special Sentinel-type units, so if the objective becomes exposed or is stolen, these units will be ordered to reinforce the objectives. There are twelve enemies in total, therefore each surviving AI-controlled unit is worth 7.5 points for the AI.

The third level requires the player to properly coordinate the action of all his units. The challenges, like simultaneously facing two Sentinels or Wardens, were designed to be too difficult for one or even two units to overcome.

Because the player will probably need all three of his units to prevail in the final skirmish with Chasers and Special Chasers, he could consider sending a less damaged unit to draw the enemy fire, while shooting with his remaining units to defeat the foe. Such tactics can increase the chances of all units surviving the final battle. There are fifteen enemies in total, hence for every surviving AI-controlled unit the AI is awarded 6 points.

4. The experiments

The aim of the conducted experiments was to verify how effective the implemented artificial intelligence methods were in creating challenges and interesting experiences for the player.

The main objective of game AI is to build a compelling experience [10]. Therefore, to properly research artificial intelligence methods for strategy computer games it was necessary to verify if they fulfill this task satisfactorily.

Because the experimental game is a game of strategy, the experience created by it should be based on a challenge requiring the human player to think through and plan his actions. Chance and manual skills should play a minor role in achieving victory [3]. Different types of enemies should be consistent in their behaviors to allow the player to learn their patterns and use this knowledge to build his strategy. The more experienced player is the more he should be aware of these facts and use them to his advantage.

The game was presented to several human players with varying experience in playing computer games. The participants did not know any details about the game (like controls or types of enemies) before playing it. All of their knowledge was supposed to come from the game itself, especially the tutorial. The exception was the scoring system—they were aware of the rules for awarding points. The participants played all the game levels and their actions in the game were observed to examine what strategies were used by them. After each trial to beat the level, both the player's and AI's scores were noted. Additionally, when the player finished the level, the number of tries he needed to do so was documented.

Below, the details of the experiment participation of all the players are presented. The participants are sorted by their estimated level of proficiency in playing computer games in ascending order.

Player 1 has almost no experience with computer games. He played only several times for a few minutes during his lifetime.

On the level 1 (see Table 1) the player was learning the rules of the game and how to control the game for quite a long time. He did not coordinate the actions of his units, always used one at a time. He was slow to react to danger and lost two units in an encounter with Chasers, but could still beat the game level on the first try.

Table 1. Player 1 scores.

Player	Level 1	Level 2			Level 3	
	1	1	2	3	1	2
Player 1	270	130	145	285	50	225
AI	0	177	125	0	196	0

On the level 2 (see Table 1) the player kept using only one unit at a time, and was overwhelmed by Chasers and Special Chasers on his first trials. However, his slow reaction time somehow worked to his advantage. Because he moved his units slowly, he could place them slightly out of range of enemy units, but still allowing them to hurt the enemy. He very patiently kept shooting from this long range until every enemy unit was defeated.

In the final level the player did not change his strategy—he gave orders to units individually. In his second trial he utilized the cover mechanic. That, combined also with patient shooting the enemy from the long range, was enough for him to achieve victory, but with an unimpressive score of 225—barely higher than the lowest possible winning score of 210 (see Table 1).

The participant was impressed by the different behaviors of enemy units and found that they appeared intelligent. He said he used a strategy of keeping his distance and distributing tasks among his units to win.

Player 2 is not currently playing computer games. Several years ago, he did this for two to four hours a week. During this time, he focused on simple action games that do not require strategic thinking and planning.

The player did not read the rules written in the tutorial carefully, instead he attacked the enemy using only one unit at a time, which resulted in a loss. In his second attempt, he read the rules and kept his units together, which allowed him to defeat his enemies (see Table 2).

Table 2. Player 2 scores.

Player	Level 1		Level 2						Level 3		
	1	2	1	2	3	4	5	6	1	2	3
Player 2	80	300	80	80	80	160	65	270	50	65	225
AI	182	0	190	190	175	125	212	0	226	190	0

In the second level, the player tried various approaches, such as focusing first on defeating enemy forces and then collecting objectives, or using Sentry-type units to lure Special Chasers and defeat them one at a time. However, the player still struggled to be cautious and tried to get through the level quickly, losing five times (see Table 2).

At the last level, the player still had problems with the coordination of the entire team, which was required at this level. In his third attempt, he kept his units in a tight formation and managed to win the level, but with an unremarkable score of 225 points (see Table 2).

The participant said that the enemy units appear to be smart to some degree because they tried different tactics, like aggressively chasing the player, running to cover or calling for help. The player admitted that some strategic planning is required to perform well in the game, that it is important to keep the units together and to keep distance from the enemy units because their accuracy increases with proximity to the target.

Player 3 currently plays computer games for about two hours a month. A few years ago, he played two to five hours a week. On the tutorial level the player carefully kept his units together, but was surprised by the aggressive behavior of the Chasers and lost a unit while fighting them (see Table 3).

The player approached the second level cautiously, but in his first attempts he was still overrun by alarmed Chasers and Special Chasers. Then he tried to use the cover to his advantage. This behavior, combined with his tight formation strategy, allowed him to finish the level (see Table 3).

Table 3. Player 3 scores.

Player	Level 1	Level 2				Level 3	
	1	1	2	3	4	1	2
Player 3	285	130	130	130	300	65	300
AI	0	162	155	147	7	202	0

After viewing the challenges of the last level on the first try, the player used the tactics learned on the previous level and won on the second try (see Table 3).

The participant stated that some units used clever tactics and others did not appear to be intelligent. The player said that the strategy of using cover and keeping the units in cohesive formation allowed him to win.

Player 4 plays computer games for approximately five hours a week. On the first level (see Table 4), the player took his time to get acquainted with the detailed capabilities of his units, such as their range, rate of fire and speed. He lined up his units and moved them carefully so as not to break the formation. He went up to each enemy with the entire team and ensured victory.

Table 4. Player 4 scores.

Player	Level 1	Level 2	Level 3
Player 4	300	300	300
AI	0	7	0

While playing at level 2 (see Table 4), the player discovered that his units had a slightly greater range than enemy units. If a player's unit is placed just outside the enemy's spotting range, it can damage the enemy unit without fighting. The player was careful and moved his units very slowly to always keep this perfect distance and use it to defeat the AI.

In the last level (see Table 4) the player used the same strategy as in the previous level, using the greater range of his units and completing the level flawlessly.

The participant believes that some elements of enemy units' behavior require corrections, for example they do not react to fire and thus the player easily beats the entire game. The player focused on his strategy of keeping his units out of the enemy's reach and that was enough to win.

Player 5 plays computer games for about ten hours a week, but as he is the youngest participant his overall gaming experience may be lower compared to other players. In the first level (see Table 5), the player quickly learned the controls and defeated the enemy while keeping his units together.

Table 5. Player 5 scores.

Player	Level 1	Level 2	Level 3
Player 5	300	270	300
AI	0	0	0

At level 2 (see Table 5), the player focused on combining scattered units. He had no problems defeating the enemy forces, but he gained overconfidence and carelessly rushed to the third target, losing two units.

During the game at level 3 (see Table 5), the player had no difficulty coordinating the maneuvers of his entire team, holding the units together and making them use cover.

The player stated that the game was too simple to tell if the enemy's behavior was rational—he ended the game before the characters could appear to be intelligent. The competitor agreed that he had to use formation and cover strategies to win the game.

The conducted experiments show that the game was rather easy to complete, with barely over half of the participants being forced to give more than one try to finish any of the game's levels. Nevertheless, even the most proficient players had to approach the game with caution and use strategic planning to be able to win. Different strategies were utilized—using cover, engaging enemy units one by one, staying in formation, staying out of the enemy's range—with varying degrees of success.

The artificial intelligence methods used have been successful in creating the right strategy game experience—players were forced to consider and plan their actions and developed various strategies to win the game.

5. Conclusions

The main goal of this paper was to examine the various techniques used to implement artificial intelligence modules for strategy computer games. The fundamental requirements for artificial intelligence in strategy games, namely: pathfinding, decision-making, tactical and strategic reasoning, and group movement (for real-time strategy games) were presented. Selected AI methods for fulfilling these requirements were also discussed. The exemplary real-time strategy game, which was developed with the use of selected AI methods was described and experimentally verified. The game was presented to several players with differing experience with computer games.

As expected, the game AI was harder to beat for less proficient players, while remaining relatively easy for the experienced ones. However, what is more important, this experiment showed, that the chosen methods, even in a simple game, were successful in creating a challenge, which required all the players, even the most experienced ones, to plan their actions and develop strategies to beat the game AI. Just like in a proper strategy game, luck and manual skills were not enough to win.

The presented AI methods, which were described in detail and verified in the experimental game, are still only a small part of available solutions, which can be used to create AI modules for strategy computer games. The experimental game, while sufficiently exploiting implemented methods of artificial intelligence, is barely complex enough to require strategic thinking. It requires some degree of planning, but it is possible to additionally enhance this strategic experience by adding some new features like resources, possibilities of creating new units, and player's units with distinct characteristics. Also, using adaptive approaches [13, 4] would possibly lead to a more demanding strategies used by the AI module.

Acknowledgements

This research was partially supported by the funds assigned to AGH UST by the Polish Ministry of Science and Higher Education.

References

[1] Barrera, R., Kyaw, A.S., C., P., Swe, T.N., 2015. Unity AI Game Programming. 2 ed., Packt Publishing.
[2] Buckland, M., 2005. Programming Game AI by Example. Wordware Publishing.
[3] Dixit, A., Skeath, S., Reiley, D.J., 2014. Games of Strategy. 4 ed., W. W. Norton.
[4] Dreżewski, R., Klęczar, M., 2018. Artificial intelligence techniques for the Puerto Rico strategy game, in: Jezic, G., Kusek, M., Chen-Burger, Y.H.J., Howlett, R.J., Jain, L.C. (Eds.), Agent and Multi-Agent Systems: Technology and Applications. 11th KES International Conference, KES-AMSTA 2017 Vilamoura, Algarve, Portugal, June 2017 Proceedings, Springer International Publishing, Cham, Switzerland. pp. 77–87.
[5] Graham, R., McCabe, H., Sheridan, S., 2003. Pathfinding in computer games. ITB Journal .
[6] Hu, J., Gen Wan, W., Yu, X., 2012. A pathfinding algorithm in real-time strategy game based on unity3d, IEEE Audio, Language and Image Processing (ICALIP) International Conference.
[7] Lecky-Thompson, G., 2008. AI and Artificial Life in Video Games. Cengage Learning.
[8] Millington, I., 2019. AI for Games. 3 ed., CRC Press.
[9] Rabin, S., 2012. AI Game Programming Wisdom. Charles River Media.
[10] Rabin, S. (Ed.), 2013. Game AI Pro: Collected Wisdom of Game AI Professionals. CRC Press.
[11] Schwab, B., 2004. AI Game Engine Programming. Charles River Media.
[12] Sánchez-Crespo Dalmau, D., 2003. Core Techniques and Algorithms in Game Programming. New Riders Games.
[13] Wilisowski, Ł., Dreżewski, R., 2015. The application of co-evolutionary genetic programming and TD(1) reinforcement learning in large-scale strategy game VCMI, in: Jezic, G., Howlett, R.J., Jain, L.C. (Eds.), Agent and Multi-Agent Systems: Technologies and Applications. 9th KES International Conference, KES-AMSTA 2015 Sorrento, Italy, June 2015, Proceedings, Springer International Publishing, Cham, Switzerland. pp. 81–93.