Granular Games in Real-Time Environment

Maciej Świechowski
Silver Bullet Labs
Warsaw, Poland
m.swiechowski@mini.pw.edu.pl

Dominik Ślęzak
Institute of Informatics, University of Warsaw
Warsaw, Poland
slezak@mimuw.edu.pl

***Abstract*—We propose a new approach to assist computer game creators in introducing AI agent-players into their games. We point out that traditional methods, such as Monte Carlo Tree Search (MCTS), may not provide creators with good interfaces to embed the required AI elements because of too fine-grained space of (often loosely defined) game states. Thus, we suggest to follow the paradigms of information granulation and re-define states/actions at a higher level of abstraction, so the MCTS algorithms can operate on more general concepts, which reflect the creators' domain knowledge. In our approach, the game developers are responsible for specification of mechanisms behind particular high-level states/actions from the perspective of "real world of the game". Meanwhile, the MCTS routines take advantage of the fact that many unique sequences of fine-grained actions become to fall into the same clusters reflecting information granules corresponding to the introduced concepts.**

***Index Terms*—Real-Time Video Games, Monte Carlo Tree Search, Information Granulation, Simplified Games**

I. Introduction

There has been a lot of successful research done in the area of utilizing the elements of AI in making computer games more competitive and interesting for human players [1]. Still, it remains quite challenging to develop tools and libraries that would make the process of introducing AI-based components easier for professional game creators [2]. The current most popular methodologies assume that game creators implement behaviors of "intelligent" game-playing agents in quite a hand-crafted way, without the ability to introduce multiple layers of abstraction and let these AI agents dynamically search for optimal actions at the level more comparable to the way of human reasoning. On the other hand, we believe that this kind of abstraction – a kind of game simplification/granulation – would allow game creators for introducing AI agents that are both, more tractable from a creator's perspective and more "human-like" from the viewpoint of an average player.

In this paper, we attempt to make a step toward establishing a more intuitive AI-related library for developers of real-time games. We proceed with the idea of information granulation [3], that is, clustering together both, states and (sequences of) actions, so it becomes possible to operate with higher-level concepts describing what can actually happen during a given game play. We actually believe that this kind of approach can be useful for multiple purposes, including not only game developers but also end users – i.e. casual human players – who may wish to seek for summaries of their game experiences in a simplified, higher-level language [4].

This work was co-financed by EU Smart Growth Operational Programme 2014-2020 under GameINN project POIR.01.02.00-00-0150/16.

As our proposal refers to adapting the aspects of information granulation into real-time games, we call it, for short, the *granular games* approach. It is based on the mechanisms of Monte Carlo Tree Search (MCTS) [5], [6], though it re-defines – and actually highly simplifies the space of states/actions that the MCTS algorithm needs to work with in order to make it computationally tractable. Representation of a game in a simplified (abstract) fashion needs to preserve its nature and dynamics, i.e., some strategic/tactical decisions that can be found at the simplified – higher – level should be easily transferable to the level of the actual game. Thus, if computer games are said to be models of real or fictional worlds, then we go one step deeper and propose a model of a game. Such a model can be used to derive useful knowledge about the game, although our main goal in this paper is to utilize it to find quasi-optimal simplified (abstract) actions for an agent in the game that is controlled by the computer.

There is a whole variety of ways how granular approaches can be applied to the MCTS algorithms [7], [8]. Herein, from the perspective of iteratively performed MCTS steps – namely, selection, expansion, simulation and back-propagation – one of the most pronounced distinctions is whether the simulation phase is fine-grained or coarse-grained.

1) **Fine-grained simulation:** The whole game logics such as determining legal actions, applying actions, updating states, etc., are performed at the finest-grained level. As an effect, a game simulation and a normal game use the same mechanism.
2) **Coarse-grained simulation:** One can simulate games using abstract concepts only. This variant is usually much more difficult to implement because the simulator must operate on a representation that the game has not been designed originally for. For example, a course grained action can be "attack the opposing army". The resulting state is usually one of many possible states, hence, coarse-grained simulations are usually based on probability and sampling from a space of possible realizations.

Our approach employs three granulation levels. First, there is an actual game. Our model does not perform simulation on this finest-grained level, as it would be computationally infeasible. Next, there is a simplified game simulation level, where the simulations are actually performed. Finally, there is

a tree level, which contains abstract states and actions only for the purpose of storing the statistics. This is the coarsest-grained level. As for now, in spite of working with three above levels, we assume that the human input will be taken into account rather only at the first level of abstraction. However, this assumption can change in the future.

The paper is organized as follows. In Section II, we recall the MCTS algorithm. In Section III, we outline our motivation for putting together the ideas of information granulation and MCTS, particularly from the perspective of real-time video games. In Section IV, we introduce the six-component model that we propose as the means for supporting game developers. In Section V, we show how to implement our model so it operate efficiently within the MCTS framework, including communication between the aforementioned three layers of granularity. Section VI concludes our research. Although this work should be regarded as a position paper, we enrich it with the feedback obtained from the first iteration of prototypes that we have already implemented.

II. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [5] is an increasingly popular method for developing game-playing AI agents. It is based on an iterative tree-search without the need for any heuristic evaluation function. Instead, a random sampling is performed through simulations to the end of the game. The algorithm gradually builds a tree in the computer's memory with statistics of the performed simulations (see Figure 1). The four phases of MCTS are as follows:

1) **Selection:** The algorithm starts from the root node and searches the tree down by choosing subsequent children nodes. The child node at each node down the path is chosen according to the so-called selection policy. The selection phase ends when there is no child node to choose, i.e., a leaf node has been reached.
2) **Expansion:** One of possible actions is applied to a node selected in the previous step and the tree is grown by adding a child node representing the resulting state.
3) **Simulation:** The algorithm starts from the new node and performs a complete game simulation, i.e., reaching a terminal state. This phase is done outside the game-tree and no nodes are added to it. Once the simulation reaches the terminal state, the obtained goals (outcomes) of each player are fetched.
4) **Backpropagation:** The statistics are recalculated inside all nodes along the path from the root to the leaf (containing the starting state for the simulation) in the game tree. They include the average scores of each player and the number of visits to a node. An average score is computed as the total score achieved in iterations going through a particular node divided by the number of visits.

The selection formula is designed to keep a balance between exploration and exploitation. Typically, the Upper Confidence Bounds Applied to Trees (UCT) algorithm is used [6]:

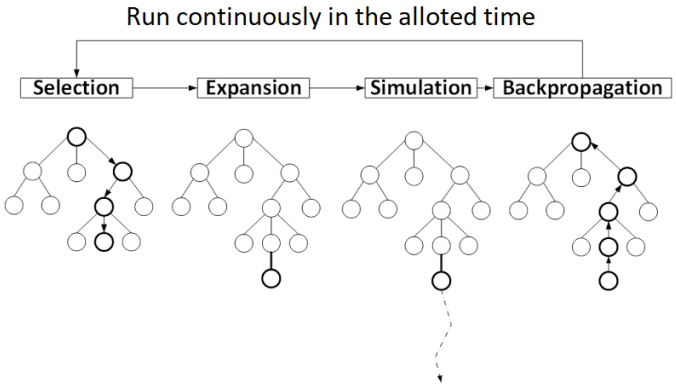

Fig. 1. Depiction of four phases, which comprise the MCTS algorithm.

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln[N(s)]}{N(s,a)}} \right\} \qquad (1)$$

where $A(s)$ is a set of actions available in state s, $Q(s,a)$ is the average result of playing action a in state s in the simulations performed so far, $N(s)$ – a number of times state s has been visited in previous simulations and $N(s,a)$ – a number of times action a has been sampled in s in previous simulations. Constant C controls the aforementioned balance. It has to be tuned, but provided that scores of games are confined to the $[0,1]$ interval, in which 0 denotes the loss and 1 denotes the win, the theoretical optimal value is $\sqrt{2}$.

The MCTS method was first successfully applied to Go [9]. Nowadays, it is the method of choice in combinatorial games such as Hex [10], Othello [11], Arimaa [12], Havannah [13], Lines of Actions [14], etc. Its universal nature makes it the state-of-the-art method for universal domains such as General Game Playing (GGP) [15] and General Video Game Playing (GVGP) [16]. MCTS is also applied outside the realm of games, e.g., in scheduling and real-life simulations.

III. Motivation for Granular Approach

Despite the advantages over traditional approaches, MCTS is still just a tree search algorithm and its effectiveness is heavily dependent on both the size and structure of the tree. If a considered problem (game) contains intractable number of possible states, then the statistical evidence gathered through iterations of the algorithm might be not confident enough due to low number of samples. The algorithm has to repeatedly sample states in order to empirically learn their quality, which is not always feasible. This is quite analogous to challenges known, e.g., from the process mining, whereby clustering similar sequences of states into more general trends may help from both, predictive and descriptive viewpoints [17].

Another detrimental factor for the MCTS performance is huge "tactical complexity" of a game, which often is reflected in existence of sequences of actions that lead to an expected positive outcome while even slight deviation from the optimal path leads to a negative outcome. This kind of complexity

suggests that game modeling – from both, designing and learning perspectives – might be better conducted in a hierarchical fashion, whereby sequences of basic actions are represented as higher-level actions that MCTS could work with. For both above reasons – complexity and sparsity of sequences of states and actions – we intend to follow in our approach the idea of information granulation that, according to Lotfi A. Zadeh, *plays a key role in implementation of the strategy of divide-and-conquer in human problem-solving* [18].

Below we discuss several properties that make developing MCTS-based game-playing AI agents in classical combinatorial games and real-time video games a different problem. One can see that, herein, hierarchical and granular approaches may be quite useful. Certainly, we do not mean that introducing higher-level concepts and actions would not be beneficial in combinatorial games. Nevertheless, we focus on real-time environments as they are a direct practical inspiration for us to consider the idea of granular games.

1) **Huge combinatorial complexity:** The main problem already introduced in this section. Significant complexity usually stems from continuous (as opposed to discrete) nature of a game. Both time and space can be continuous. For instance, in a real-time strategy game [19], a player may control a large number of units on a 2D or 3D map, so the possible combinations of each unit's movement make up for the number of possible actions and, therefore, achievable states. This alone can lead to millions of combination, not to mention units' special actions (such as attack), new units construction, handling economy, etc. From many perspectives, such challenges are comparable to real-world simulation problems, e.g., in the area of what-if analysis in decision support systems [20].
2) **Little computational budget:** In commercial real-time games, the AI module has little time available to perform computations. This is mainly due to the fact that there are many other aspects such as 3D graphics, physics, game logic, network management (in online multi-player games) and handling input that require processing power.
3) **Computational scalability:** The algorithm for AI has to not only be fast, but also must scale well with properties of the game environment such as the number of players or the size of the world. These are parameters that are usually prototyped and adjusted by game designers. The AI should not suddenly stop working (or at least not too quickly) due to increased game complexity.
4) **Lack of formal structure:** This makes representing and comparing states and actions in a game a hard problem. In commercial games, there is often no object in the architecture that represents state of the game in a structured form. The theoretical game-state can be considered as a composition of internal states of some objects that are in the game. This challenge seems to correspond to the idea of information granulation and operations on complex objects, especially given the fact that information about game states is often incomplete [3].
5) **Configurable and predictable AI:** The method should be easily reconfigurable if either game mechanics are changed or game creators decide that the AI should act differently. This aspect is specially important from the perspective of our project related to supporting game creators while introducing game-playing agents. However, predictability of AI is certainly a wider topic.
6) **Self-explanatory AI:** For verification needs, a human should be able to analyze why the AI agents act a certain way. Therefore, a representation with intractable number of nodes is not convenient. It is much easier to analyze compact representations. This aspect corresponds also to another game-related project that we are developing, namely, a support system that advises players how to improve their skills [4]. The ability to explain advices at a level that is understandable for human players is actually quite analogous to the ability to explain AI agent behaviors to game creators – we believe that both these tasks require a kind of conceptual language that describes a given game in a simplified, hierarchical way.

To address the above challenges and requirements, we are going to rely on approximate reasoning methods using an abstract model of a game. The results of such reasoning can be position evaluation in some higher-level representation of a game, which can be translated to a useful information in the actual game. If there exist certain properties, based on which states can be grouped into clusters invariant with respect to the outcome for the players, then such groups could be regarded as nodes in the game tree and the statistics could be gathered for such state groups rather than concrete states. Similar approaches have been proposed in the literature in works such as [7], [8]. However, our proposal is to extend the idea of clustering similar sequences of similar states onto a more complete framework assuming that the obtained clusters/granules correspond to some higher-level game-related concepts and letting the AI agents transform those higher-level concepts into "real" game actions.

This kind of approach seems to be analogous to other applications requiring dealing with complex spatio-temporal concepts and objects. In particular, the process of identifying those lower- and higher-level components usually requires involvement of domain knowledge about a given problem [21], [22]. The domain knowledge (on a domain level), which is often expressed in natural language, must be next linked with the system level (here based on MCTS). In the next section, we will show that our domain level is represented by boolean sentences that describe certain facts about the game but written in form of functions in a programming language.

IV. Granular MCTS Games: The Model

As already outlined, our idea is to help the MCTS algorithm by representing a complex game in a simplified (abstract) fashion. This kind of approach could be useful in many aspects of game analytics [4], although in this particular paper we concentrate mainly on supporting game creators in embedding AI players into the designed environments. This way, we

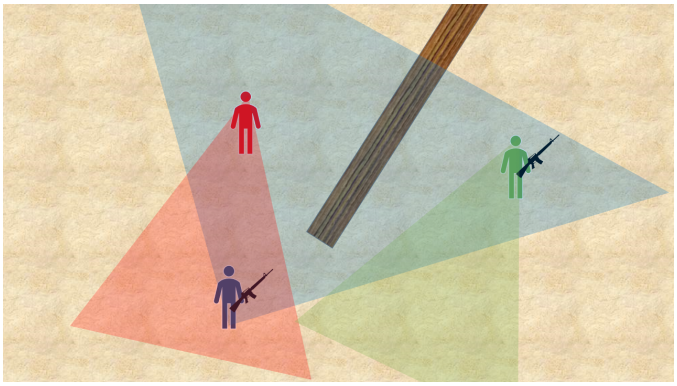

Fig. 2. A sample simplified game, which features three soldiers: blue, red and green from the left, respectively. Two of them are equipped with a gun and they are ready to shoot. The semi-transparent triangles indicate visibility range of the soldiers. We refer to Table I for exemplar relations defined in this game and their current states of truth.

TABLE I
RELATIONS DEFINED FOR THE GAME SHOWN IN FIGURE 2. VALUES IN THE SECOND COLUMN DENOTE THEIR CURRENT STATES OF TRUTH. THE LAST ROW SHOWS ABSTRACT STATE ENCODING (SEE ALSO FIGURE 3).

Boolean Relation	**Current Result**
Red Soldier can see Blue Soldier	1
Red Soldier can see Green Soldier	0
Blue Soldier can see Red Soldier	1
Blue Soldier can see Green Soldier	1
Green Soldier can see Red Soldier	0
Green Soldier can see Blue Soldier	0
Red Soldier able and ready to shoot	0
Blue Soldier able and ready to shoot	1
Green Soldier able and ready to shoot	1
Red Soldier is alive	1
Blue Soldier is alive	1
Green Soldier is alive	1
Binary encoding	101100011111
Decimal encoding	2847

continue our research related to MCTS-based game-playing AI agents [2], now extended toward real-time games. Particularly, we focus on the problem of action selection.

The following components of our model are based on prototypes that we have implemented up to now. The idea of expressing humans' needs as domain parameters, using a model to work with those parameters and finally synthesizing results follows the principles of Kansei engineering [23]. Also, the idea of composing an action based on the MCTS-driven guidelines formulated at a simplified level could be referred to the paradigms of computing with words [24].

1) **Relations** represent logical sentences about the game, i.e., statements that are *true* or *false*. A game developer provides our engine with a "check function" for each defined relation, which is a lambda expression that returns a boolean value. It operates on any number of arguments, which are either of type *FreeGameObject* (see below) or are constant throughout the game. Figure 2 depicts a simple game and Table I contains exemplar relations defined for this game, as well as results of the check functions. Because checking the state of relation is a common operation in a model, we propose some lazy-evaluation mechanism in the form of acyclic dependency graph. A relation is *true* if (1) its check function returns *true*, (2) all positive prerequisite relations are *true* and (3) all negative prerequisite relations are *false*. The positive and negative prerequisites are optional sets of relations, i.e., they can (and often will) be empty. Such boolean functions encourage game creators to pick only the most significant features of the game from the decision-making perspective. They encourage binary thinking about the game-state – each relation separates it into two areas. While it is natural for humans to provide single conditions of such a separation, it is not obvious how combined separations will interplay. This is, however, the reason why computational models are employed. Another useful feature of boolean relations is a natural and predictable way of controlling the complexity of the model. Finally, properly defined "check functions" can be reused among various games. It is worthwhile noticing that relations defined by game creators imply certain clustering of states. We expect these clusters to represent a "similar" situation from the decision point of view. The desirable property is as follows: if we took any specific fine-grained state that belongs to a cluster, then the optimal actions to be taken by the agents (by means of some measure such as game creator's expectancy) are the same or at least similar no matter how we chose the specific state. Looking at this from the actions' perspective – the actions should be robust relative to clusters. However, it is possible for an action to be optimal in many clusters and it is not a problem for the method. Given the example from Figure 2 and Table I, let us consider the action of Red Soldier shooting Blue Soldier. Such an action could be optimal when both soldiers are alive, Red Soldier can shoot, Red Soldier can see Blue Soldier while Blue Soldier cannot see Red Soldier. In such a case, even if we considered more detailed information about the current game-state, no other action should be optimal.
2) **FreeGameObjects** are any objects that relations' check functions may operate on. Their states together represent state of the game for simulation purposes. The only requirement on the structure of FreeGameObjects is that they need to implement a *reset()* function that resets their state to the starting one in the game. The MCTS algorithm will take advantage of this function.
3) **Agents** are objects that perform actions in the game. Each agent is provided with a function that determines available actions based on the relations, which are true. The MCTS algorithm will be selecting actions for agents in the selection and simulation phases. In the model, each agent also contains a team/player identifier. It is used only for backpropagation purposes: agents that share a team/player also share the score. For implementation

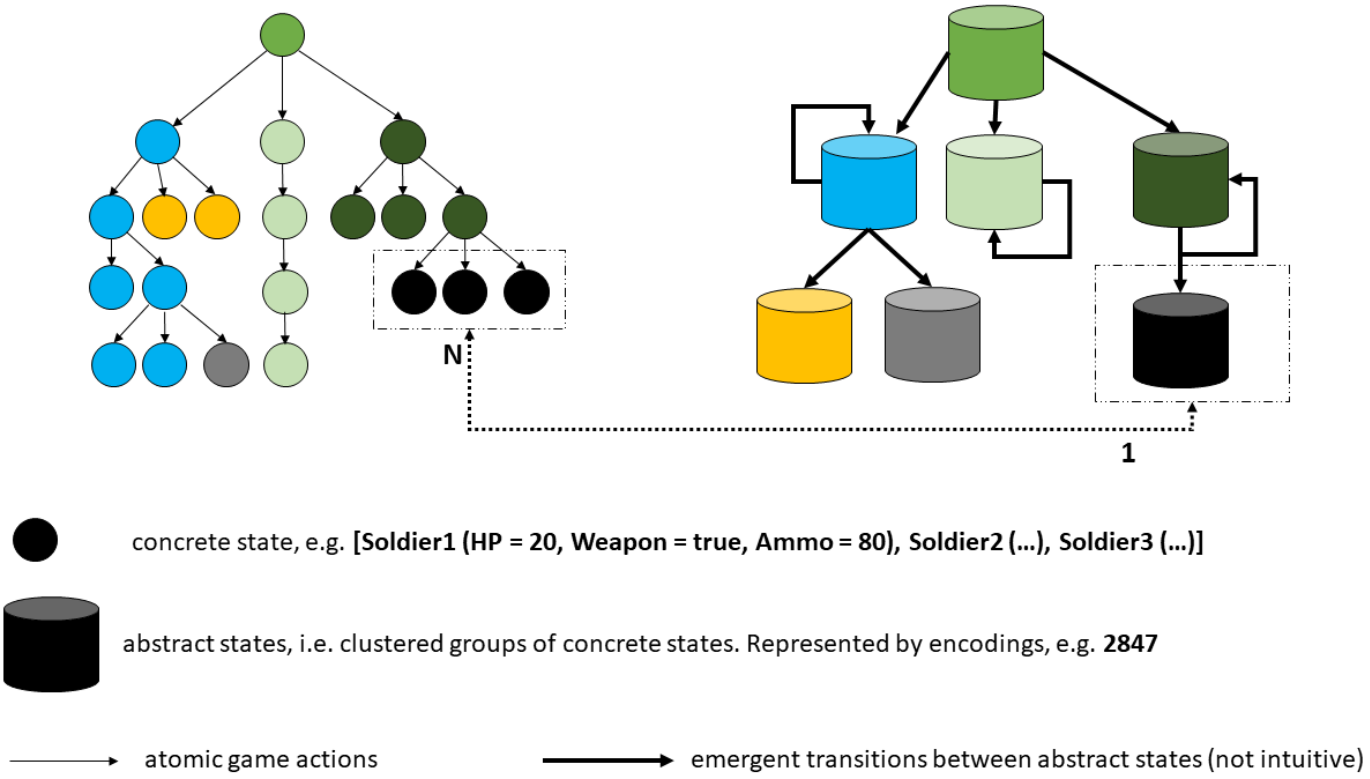

Fig. 3. Comparison of classical MCTS (on the left) with granular MCTS (on the right).

reasons, agents are assumed to be immutable from the game creator's perspective. However, it is possible to create both FreeGameObject and an Agent representing the same concept in the game.

4) **Actions** are prepared by game developers by inheriting the base action class. Actions are taken by Agents and can modify FreeGameObjects or themselves. In the proposed implementation, an action consists of three methods: *GetRemainingTime()* – the estimated time to complete; *Update(deltaTime)* – possible place to apply partial result and/or internal state of the action and *OnCompleted()*. We explain the concept of time in the model in Section V-A.
5) **Terminal function** checks whether the simplified game has reached the end. Compared to the actual game, this might happen after some important event in the game occurs. For example, a simplified game might represent a single battle in a strategy game.
6) **Score function** assigns a numerical score to each player or team of players defined in the game. The higher the score, the better for the player.

V. Granular MCTS Games: The Method

The model that was introduced in the previous section serves as an input to the algorithms that are aimed at finding the optimal action policy. In case of real-time video games, the complete optimal policy is not usually required. Instead, the problem simplifies to finding the next action to be undertaken by each agent. Below, we focus on challenges that are unique to real-time games and do not exist in combinatorial games – continuous actions with non-instant effects possible.

A. Continuous Actions

To cope with continuous actions with non-instant effects, we have made a few assumptions. First of all, we introduced virtual time to the MCTS algorithm. Actions have time to completion measured in these virtual time units. Each agent updates its active action, i.e., the currently chosen action that is not yet completed. At the start and once the agent's action is completed, it is required to choose an action among the available ones. The choice is driven by the MCTS algorithm. Second, whenever a simulation using the model is performed, it loops over all active actions (one per each agent) and calculates the minimum time of completion, i.e., the smallest time needed for any action to finish. The simulation advances by that time and each action is notified about the elapsed time commonly denoted to as "delta time" in games. Upon this notification, an action may *update* or *finish* and it is guaranteed that at least one action will be finished, because this is how the delta time was calculated. Both the *update* or *finish* methods of any action may alter the game state, however, the game state cannot be affected in any other place. This is one of our simplifications and design choices. As a result, the rule of thumb is that actions should be implemented in a generic enough way to apply its effects either upon termination or elapsed time. We leave the choice to game creators as some actions such as movement might require partial updates

whereas the others such as pushing a button might have just an instant effect on termination. However, actions cannot report too long time to completion if legality conditions of continuing the action might change often. This problem is common to physics engines [25], when the simulation step is too long and it skips through important events that would happen between two steps. Thus, we suggest to provide game creators with a mechanism to control the simulation step dynamically. We achieve this by allowing an action to change its time to completion in the *update* method. For instance, if an action's time left equals 0, the action has one more chance to increase it. Only if the action does not use this opportunity, it will be terminated. This mechanism allows to tune the simulation step span dynamically based on arbitrary game situation.

Figure 4 illustrates the idea of simulating to the nearest action expected to complete. Whenever a simulation is paused, which is exactly when an action is expected to complete, all relations are checked and the current state is captured.

B. State Capturing

This is the process of generating an abstract state in its formalized representation based on actual unstructured states of free game objects. The abstract state is a vector of 32-bit integer numbers constructed as follows. First, the state of satisfaction of each relation is determined using the underlying "check functions" and optional additional constraints – positive and negative predecessors. Let n denote the number of relations and $r_i \in \{0, 1\}$ denote the $truth$ value for i-th relation. The state of all relations can be represented by:

$$R = [r_0, r_1, r_2, ..., r_{n-2}, r_{n-1}]$$

Now, let us introduce the encoding to decimal numbers. As we encode the elements from R, hence the symbol deR:

$$deR(i, j) = \sum_{k=0}^{j-1} (r_{i+k} * 2^k); j \geqslant i$$

Finally, the abstract state is encoded as a vector of 32-bit numbers in such way, to cover all relations. For instance, if there are up to 32 relations, this vector will have one element. If there are more than 32 relations but less than 64, there will be two elements in the vector:

$$S = [deR(0, 31), deR(32, 61), ..., deR(32 * \left\lceil \frac{n}{32} \right\rceil - 32, n)]$$

Such vectors are stored in the so-called transposition table (TT) [26]. One of the most important properties of our approach is that not only it induces a certain way of simplified thinking about the game, but also it allows to implement the equality check (*equals()*) and hashing (*hash()*) functions, respectively, in an automatic manner, completely hidden from the game creators. These two functions have been often problematic, counterintuitive and inefficient for implementation. They are, however, crucial to fast retrieval and insertion of states to the transposition table. In our approach, both can be performed in the amortized $O(1)$ time.

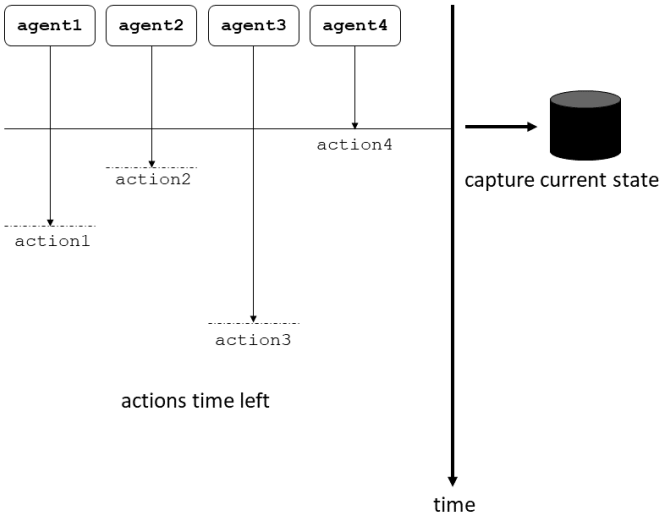

Fig. 4. The simulation advances to the action that is next to complete. Each time an action is completed, the abstract state representation is captured.

C. MCTS Iteration

Both the **selection** and **Monte Carlo** phases perform a game simulation because it is not possible to apply actions directly on the MCTS tree level without simulating them. This is a different scenario than a classical usage of MCTS, where the selection phase could start from an arbitrary node (representing state in the game), traverse an edge and proceed to a children node instead of explicitly applying the action represented by this edge. Traversing an edge and applying an action is equivalent, if and only if nodes contain full information required to perform the game logic and actions are deterministic. This is not possible in the proposed granular approach, because nodes in the tree store only an abstract encoding that does not allow for performing full game logic such as determining available actions. Abstract encoding is just a generalized static snapshot of the game-state. However, there are granular models that allow for performing search on hierarchical granular states, e.g. [27].

Although both the **selection** and **simulation** phases simulate the game in order to proceed, only the former one captures the consecutive visited states. A pseudocode of a single MCTS iteration is shown in Figure 5 and discussed below:

1) **chooseActionsForAgents:** For each agent that does not have an action chosen at the moment, one is selected according to the given policy.
2) **selectionPolicy:** The formula responsible to select an action based on its statistics so far in the selection phase. We use the UCT method from Eq.1.
3) **simulationStep:** The simulation advances by *deltaTime* to the nearest action's expected time of completion. All the game logic is applied here, the state is updated.
4) **capture:** The abstract state representation is generated according to the description in Section V-B.
5) **tt:** This variable denotes transposition table that stores all

abstract state representations visited in simulation phases of the MCTS algorithm.

6) **selectionFinished:** The selection phase is finished once such a state is captured which has not existed before in transposition table.
7) **updateHistory:** Visited states and new chosen actions are appended to a list.
8) **terminalFunction:** Provided by the game developer. The game has ended.
9) **playoutPolicy:** Also called "default policy" – a way of choosing actions in the MCTS simulation phase. Typically, it is a uniform random distribution of actions.
10) **scoreFunction:** Provided by the game developer. Returns a vector of game's outcomes for each player (team).
11) **propagate:** A method that updates the respective statistics of actions chosen by each agent during this iteration of the MCTS algorithm. The statistics include the scores and the numbers of visits.

D. Game-MCTS-Game Translation

The approach proposed in this paper assumes the three following levels of granularity:

1. **Fine-grained level:** This is the actual game.
2. **Coarse-grained level:** Abstract simplified game.
   - 2.1 **Simulation level:** Unstructured state of all free game objects + actions. This level allows for simulating a simplified game.
   - 2.2 **Decision (statistics) level:** Encoded states by relations + actions. This level is only for storing statistics and choosing actions based on state encodings. It does not, however, allow to simulate the game by any means.

The communication from the fine-grained game level (1.) to the abstract game representation (2.1) is performed at the beginning and only once per algorithm's execution. The abstract representation is instantiated dynamically in the moment game creator chooses to. It requires the six types of elements described in Section IV to be properly set up in the game's code, what gives the most flexibility. However, automatic translation might be an interesting future research topic, somewhat comparable to the ideas take from computing with words, whereby translations between "real" and "abstract" worlds are assumed to work smoothly in both directions [28]. It is also possible to take a modular approach to define simplified games with the elements as building blocks so they can be reused each time a simplified game is to be instantiated.

The communication from the simulation (2.1.) to the decision levels (2.2.) has been described in Section V-B. In essence, the decision level takes a higher-level snapshot. The backward communication, i.e., from (2.2.) to (2.1.) is straightforward based on actions. The actions are shared among both decision and simulation layers with the same level of granularity. Based on the statistics, the MCTS algorithm chooses an action to be applied in the simulation.

Finally, when the time for decision elapses, there is communication from the decision level (2.2.) back to the actual game level (1.). At this point, the MCTS has an action candidate for each agent in the game. The action is represented in terms of the simplified model (2.), so the final task is to translate this action to the finer-grained level. We propose an interface for creators that allows for three ways of doing so:

1) **One-to-one action mapping:** If a game is simple enough, actions in the model might correspond directly to the atomic game actions. The granular model might still simplify the game in other areas such as state representation, game duration, terminal condition, etc.
2) **Specialize action:** In this variant, the idea is that the action from a simplified game is general and needs to specialized in the form of a series of atomic actions. For example: the general action might be "go to the grandmother's house". The realization might be a series of actions ["take the key", "go to the forest", "take the road left until you find a big tree", "turn right", "open the door using the key"].
3) **Achieve goal:** Herein, the action from a simplified game denotes a general thing to do – a subgoal in the game. For example, "invade the opponent" can be treated as a goal to achieve. Such a goal must be solved and translated to actions to perform by some external algorithm. In our implementation of the proposed approach, we recommend using the Utility-AI system for this task.

```
history = []
while(selectionFinished not true)
  chooseActionsForAgents(selectionPolicy)
  simulationStep(calculateDeltaTime())
  currentState = capture()
  selectionFinished = tt.exist(currentState)
  if(selectionFinished)
    tt.add(currentState)
  updateHistory()

while(terminalFunction() not true)
  deltaTime = calculateDeltaTime()
  simulationStep(deltaTime)
  chooseActionsForAgents(playoutPolicy)

scores = scoreFunction()
propagate(history, scores)
```

Fig. 5. Pseudocode of a single MCTS iteration using the granular model.

VI. Discussion

We discussed a need for granular computational models that fuel AI in games. Although there exist some notable games such as Go [1], [9], for which a more direct (non-granular, non-hierarchical) approach is proven to be successful, this is not the case in the realm of real-time video games [19].

We proposed a computational model called a granular game model, which is based on boolean relations describing the original game environments in a simplified fashion. The model is convenient for implementation and will be part of our currently developed game AI library [2]. We also introduced a modified MCTS algorithm tailored for the proposed model. The algorithm takes an abstract representation of a game, performs calculations and returns abstract actions to be taken. Those actions are translated, if needed, to a finer-grained level of detail, following the ideas of information granulation and computing with words [18], [27]. Some details of possible realizations of such translation are shown in the paper.

Our future plans include further improvements of the proposed approach, commercial implementation of the presented ideas, as well as research on multi-resolution granular models. We intend to continue comparing our methodology with other applications that take an advantage of abstracting higher-level states and actions for game modeling purposes [28]. We will also introduce analogous mechanisms of information granulation into our other research projects related to real-time game analytics and player-oriented advisory systems [4].

Actually, human players and game creators have a lot in common, as both communities seem to need intuitive "natural language interfaces" to communicate with game environments. Developing methods for such communication requires solving many challenges. On the other hand, it gains a lot of interest in different fields of science and industry. For instance, let us refer to [29]: *Causal inference requires two additional ingredients: – a science-friendly language for articulating causal knowledge, and – a mathematical machinery for processing that knowledge*; and [30]: *Tomorrow, I believe, every biologist will use computer to define their research strategy (...), and extend their experimental observations – through exploratory discovery and modeling*. With these two citations in mind, let us conclude this paper with a statement that the proposed granular games can be treated as an example of a broader trend devoted to simplified modeling and reasoning about complex phenomena in both, physical and digital worlds.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] M. Świechowski, T. Tajmajer, and A. Janusz, "Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms," in *Proceedings of IEEE CIG 2018*, 2018, pp. 445–452.

[3] A. Skowron and A. Jankowski, "Rough Sets and Interactive Granular Computing," *Fundamenta Informaticae*, vol. 147, no. 2-3, pp. 371–385, 2016.

[4] A. Janusz, D. Ślęzak, S. Stawicki, and K. Stencel, "SENSEI: An Intelligent Advisory System for the eSport Community and Casual Players," in *Proceedings of WI 2018*, 2018.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[6] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proceedings of ECML 2006*, 2006, pp. 282–293.

[7] A. Bai, S. Srivastava, and S. Russell, "Markovian State and Action Abstractions for MDPs via Hierarchical MCTS," in *Proceedings of IJCAI 2016*, 2016, pp. 3029–3037.

[8] J. Hostetler, A. Fern, and T. Dietterich, "State Aggregation in Monte Carlo Tree Search," in *Proceedings of AAAI 2014*, 2014, pp. 2446–2452.

[9] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions," *Communications of the ACM*, vol. 55, no. 3, pp. 106–113, 2012.

[10] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[11] D. Robles, P. Rohlfshagen, and S. M. Lucas, "Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search," in *Proceedings of IEEE CIG 2011*, 2011, pp. 305–312.

[12] O. Syed, "The Arimaa Challenge: From Inception to Completion," *Journal of the International Computer Games Association*, vol. 38, no. 1, pp. 3–11, 2015.

[13] F. Teytaud and O. Teytaud, "Creating an Upper-Confidence-Tree Program for Havannah," in *Proceedings of ACG 2009*, 2009, pp. 65–74.

[14] M. H. Winands, Y. Bjornsson, and J.-T. Saito, "Monte Carlo Tree Search in Lines of Action," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 239–250, 2010.

[15] M. Świechowski and J. Mańdziuk, "Self-Adaptation of Playing Strategies in General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 367–381, 2014.

[16] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General Video Game Playing," *Dagstuhl Follow-Ups*, vol. 6, 2013.

[17] S. Tsumoto, H. Iwata, S. Hirano, and Y. Tsumoto, "Similarity-based Behavior and Process Mining of Medical Practices," *Future Generation Computer Systems*, vol. 33, pp. 21–31, 2014.

[18] L. A. Zadeh, "A New Direction in AI: Toward a Computational Theory of Perceptions," *AI Magazine*, vol. 22, no. 1, p. 73, 2001.

[19] M. Buro, "Real-Time Strategy Games: A New AI Research Challenge," in *IJCAI 2003*, 2003, pp. 1534–1535.

[20] A. Krasuski, A. Jankowski, A. Skowron, and D. Ślęzak, "From Sensory Data to Decision Making: A Perspective on Supporting a Fire Commander," in *Workshop Proceedings of WI-IAT 2013*, 2013, pp. 229–236.

[21] J. G. Bazan, "Hierarchical Classifiers for Complex Spatio-temporal Concepts," in *Transactions on Rough Sets IX*. Springer, 2008, pp. 474–750.

[22] S. H. Nguyen, T. T. Nguyen, M. S. Szczuka, and H. S. Nguyen, "An Approach to Pattern Recognition Based on Hierarchical Granular Computing," *Fundamenta Informaticae*, vol. 127, no. 1-4, pp. 369–384, 2013.

[23] M. Nagamachi, "Kansei Engineering and Rough Sets Model," in *Proceedings of RSCTC 2006*, 2006, pp. 27–37.

[24] L. A. Zadeh, *Computing with Words – Principal Concepts and Ideas*. Springer, 2012.

[25] T. Erez, Y. Tassa, and E. Todorov, "Simulation Tools for Model-based Robotics: Comparison of Bullet, Havok, Mujoco, Ode and Physx," in *Proceedings of IEEE ICRA 2015*, 2015, pp. 4397–4404.

[26] A. Kishimoto and J. Schaeffer, "Transposition Table Driven Work Scheduling in Distributed Game-Tree Search," in *Proceedings of AI 2002*, 2002, pp. 56–68.

[27] J. Luo and Y. Yao, "Granular State Space Search," in *Proceedings of Canadian AI 2011*, 2011, pp. 285–290.

[28] A. Sinha, F. Fang, B. An, C. Kiekintveld, and M. Tambe, "Stackelberg Security Games: Looking Beyond a Decade of Success," in *Proceedings of IJCAI 2018*, 2018, pp. 5494–5501.

[29] J. Pearl, "Causal Inference in Statistics: An Overview," *Statistics Surveys*, vol. 3, pp. 96–146, 2009.

[30] J. C. Wooley, "Foreword," in *Computational Modeling of Genetic and Biochemical Networks (Computational Molecular Biology)*, J. M. Bower and H. Bolouri, Eds. The MIT Press, 2004.