Improve Offline Adaptation of General Video Game Playing Agents

Margarita Naryzhnyaya
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—General Video Game Playing poses as problem the creation of agents that are able to play and win unknown in advance video games. They should adapt to the environment, and use information from past experiences is crucial for them to act accordingly. Adaptation and learning are two important aspects of intelligence, therefore information about how agents perform in certain game environment is an interesting point to consider. Classification models can be trained on previously played games, and used for adjusting the playing-agent's parameters with respect to its environment, defined by features of the games.**

***Index Terms*—Games, General Video Game Playing, Monte Carlo Tree Search, Machine Learning Classification**

I. Introduction

Games have often provided practical and convenient test-bed environments to evaluate work in Artificial Intelligence (AI). For instance, contributions to planning [1] and navigation [2] was made possible by testing algorithms in environments of high and increasing complexity of games. Path-finding algorithms were able to improve thanks to simulated and realistic worlds offered by games. Some advancements of existing algorithms resulted from variants of tree search algorithms, such as the Monte Carlo Tree Search (MCTS) [3], as well as variants of the A* algorithm [4].

General intelligence, along with social intelligence and creativity, has been identified as one of the most important long-term goals of AI, and the domain of games has proposed very effective strategies to fulfill these three goals [5]. Studying General Game Playing (GGP) [6], which is about designing agents that are able to play unknown games of a large variety, is a core aspect of general AI since the focus is not narrowed to a specific game. GGP agents cannot rely on algorithms designed in advance for specific games, so the expertise must depend on the intelligence of the game player itself. Therefore, underlying technology can be used in a variety of other application areas, for instance in business [7], military simulations [8] and electronic commerce [9]. The idea of GGP was extended to video games, forming the area of General Video Game Playing (GVGP) [10]. Research work done in this area enabled some advancement in other domains of AI: in Machine Learning (ML) [11], neuro-evolution algorithms [12], and natural language processing [13].

This thesis was prepared in partial fulfilment of the requirements for the Degree of Bachelor of Science in Data Science and Artificial Intelligence, Maastricht University. Supervisor(s): Matthew Stephenson, Dennis Soemers

In GGP, and GVGP, a hyperagent, or portfolio agent [14], is an agent that can adjust its parameters, or sub-agents in some cases, which modifies its behaviour, and therefore it influences its performance. The purpose of this research is to investigate how to use the agent's environmental knowledge, i.e. certain game features, to improve its game play-ability by adjusting the parameters to the ideal values. Analysis of the MaastCTS2 [15] sub-agent's performances is performed by the mean of classification. A model is built to find some correlation between features of games and the agent's parameters. To study how to improve MaastCTS2's performance, taking into account some characteristics of the games, three research questions are posed:

1) How effective are the classification methods at predicting which parameter values give the best performance?
2) How can we use classification methods to implement a hyper-agent that selects good parameter configurations for each new game?
3) What is the performance of such a hyper-agent when used to play a wide variety of video games?

This thesis is structured as follows. Section II describes in more depth the concept of GVGP, as well as some details of the competition and the framework, with the MaastCTS2 agent. Next, in Section III, subsection III-A explains the details about how the data can be gathered from the games, the agent that is playing, and the outcome from the game-play. Subsection III-B explains the details of the models used to predict the parameters that the portfolio agent should have to play unseen games. Afterwards, in Section IV, the outcomes of the accuracy of the classification models, as well as the performance of the adapted agents are presented. After the discussion about the accuracy of the prediction models and the performance of the agents in Section V, the thesis ends with a conclusion (Section VI) about the prediction models, and what can be enhanced in future work, in Section VII.

II. Backgroud

The General Game Playing Competition (GGP) [6] challenges the participants to create agents that can effectively play a variety of previously unseen games, the majority of which are variants of existing board games, or turn-based discrete games. The General Video Game AI competition (GVGAI) [16] was also created in order to test general agents, but this time on video games.

A. *GVGAI framework*

The GVGAI framework [16] describes games via the Video Game Description Language (VGDL) [17]. The description of the entities, interactions, and termination requirements constitutes the definition of the games. A valid agent to work in this framework is the one that can select the moves in a real times fashion. The framework does not provide all the information to the agent, such as the rules of the game, the behaviours of the sprite and the termination requirements. On the other hand, the player can query its state, other sprites' positions, the game status, possible moves, potential successor states, and the history of events of collisions [16].

B. *Agents*

Variations of some algorithms implemented in different hyper-agents were noticed to show best performance. In fact, from the results of the GVGP competition, in 2014 [14], the algorithms that performed among the best at playing multiple games were those using MCTS algorithms, heuristics or Hierarchical Open-Loop Optimistic Planning algorithm, as well as some variants of those. It was also observed that some agents performed better in some games, and other agents in other games. Among them, the MaastCTS2 won the competition for Single-Player track in 2016 [18]. Since this thesis focuses on Single-Player games, this agent is suitable for this research, on top of being competitive, open-source and having many parameters that can be adjusted. This agent uses Open Loop Monte-Carlo Tree Search [19]. Many combinations of sub-agents can result from setting different parameters to the MaastCTS2. Table I lists the parameters that are tuned, and the different values that are considered, with the values of the parameters that were set for the competition of 2016 in bold.

TABLE I: MaastCTS2 parameters - Values of the parameters set for the competition in 2016 in bold

Parameters	Type	Values
Selection strategy	Categorical	**ProgressiveHistory(0.6, 1.0)**, ProgressiveHistory(0.0, 1.0), ProgressiveHistory(0.6, 0.5), ProgressiveHistory(0.9, 1.0), olUct(0.6), olUct(0.9)
Payout strategy	Categorical	**NstPlayout(10, 0.5, 7.0, 3)**, NstPlayout(10, 0.5, 9.0, 3), NstPlayout(7, 0.5, 9.0, 3) Random(10.0), Random(6.0), Random(7.0) MAST(1.0, 0.5), MAST(7.0, 0.5)
Move selection strategy	Categorical	**MaxAvgScore()**
Playout evaluation	Categorical	**GvgAiEvaluation()**
initBreadthFirst	Boolean	**true**, false
noveltyBasedPruning	Boolean	**true**, false
exploreLosses	Boolean	**true**, false
knowledgeBasedEval	Boolean	**true**, false
treeReuse	Boolean	**true**, false
treeReuseGamma	Double	**0.6**, 0.3, 0.8
maxNumSafetyChecks	Integer	**3**, 5, 9
alwaysKB	Boolean	**true**, false
noTreeReuseBFTI	Boolean	true, **false**

III. Methods

A. *Data gathering*

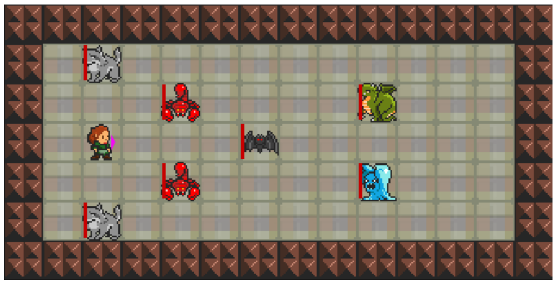
The process to gather the information about the games, and the parameters of the agent that plays them, goes as follows:

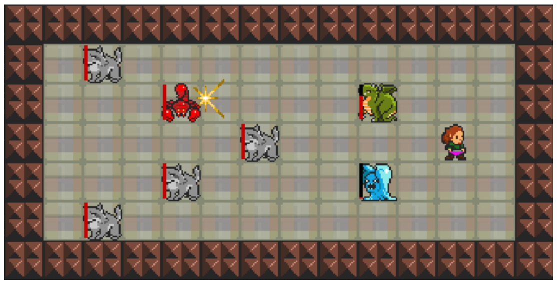
1) At the start of one game that the agent will play (the five levels of that game will be played five times) a .csv file is initialised, having as name a string with the parameters of the current agent. Each row stores information about the game played by the agent. One column stores one features of the game played, one other the name of the game and one column stores the level.
2) A string is initialised at the beginning of each game with the name of the game, the level that is being played and the features of the game, more detail follow in Section III-A1.
3) At the end of the game, the results are stored at the end of this string. The results are made up of the victory status of the player, the score and the time step.
4) For each game played, a new line is initialised and steps 2) and 3) are repeated.
5) When all the the levels of the game are played five times, the .csv file is stored.

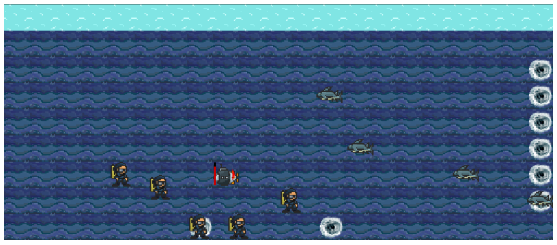
At the end, the number of files is the number of games played by the agent, containing the results that the agent played. The name for each file contains the name of the game played, and a string storing the values of the parameters of the agent that played the games. This enables to save memory space since there is not one column for each parameter, these columns are added later to avoid redundancy in the initial files. Later on, the .csv files are imported in Python and are treated as data-frame objects. All the .csv files are then concatenated into one big data-frame to perform classification and predictions, with new columns for each parameter.

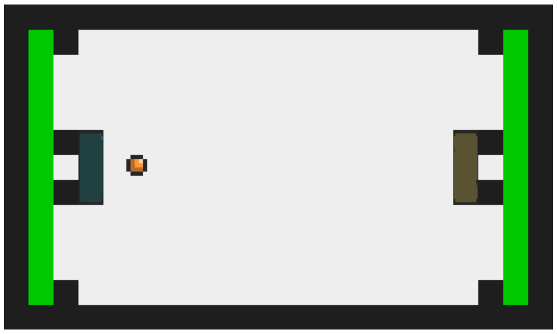
1) Level Features: The GVGAI framework provides 122 games, each one has 5 levels. Feature extraction for each of them happens at the very beginning of the one game level: one part of the features are extracted from the State Observation (given to the agent at the initial game state), the object that provides the agent with information about the game's status, and the other part from the Game Description object, which information are never given to the agent under standard competition rules. The features from the latter object are extracted for this thesis, even if under the competition conditions they are not given to the agent. Table II lists all the level features extracted. Afterwards, it is explained which features are stored directly to the .csv file, and which are used to make new columns to store the features shown in the Table II later on in Python. The .csv files are read as data frames, where the current features' columns are computed from the stored data. Among the information given to the agent, there are:

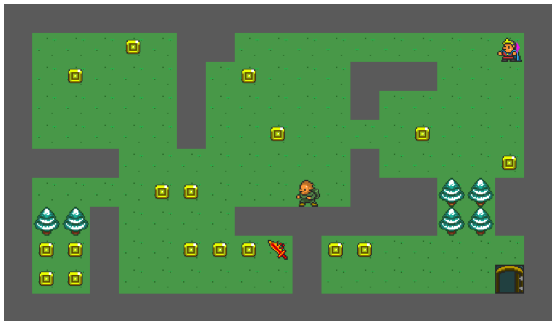
- All the features in the group Total amount, except for the *nNPC* and *nInteractions*.

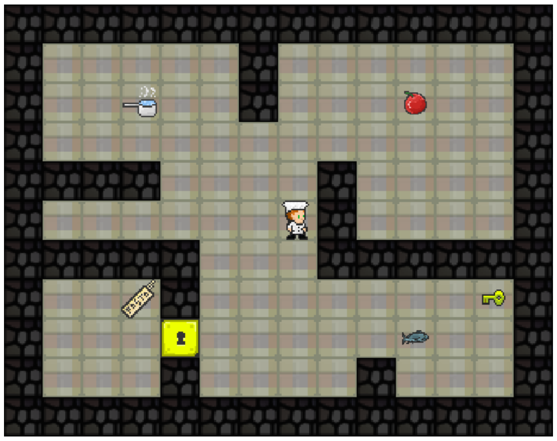

(a) Pokemon level 2: start of the game


(b) Pokemon level 2: middle of the game


(c) Sea quest level 2: 8 portals


(d) Pong level 1: available actions are up and down


(e) Overload: level 1: 18 resources


(f) Cook me pasta level 2: area of 500346.0

(g) Painter level 1: area of 28800.0

Fig. 1: Games with visualisation of features

TABLE II: Selected game features

Group	**Feature**	**Type**	Examples					
			Pokemon	Cook Me Pasta	Painter	Sea Quest	Pong	Overload
Total amount	*nNPC*	Integer	8	0	0	2	0	1
	nImmovable	Integer	41	66	8	189	96	303
	nMovable	Integer	0	4	0	0	2	0
	nPortal	Integer	0	0	0	5	0	0
	nResource	Integer	0	1	0	0	0	18
	nAction	Integer	5	4	4	5	2	5
	nInteraction	Integer	1	10	4	7	3	5
Map	*area*	Double	318402.0	500346.0	28800.0	272916.0	370656.0	368676.0
	horizontalRatio	Double	0.143	0.5	0	0.476	0.136	0.895
	verticalRatio	Double	0.429	0.455	0	0.222	0.385	0.091
	density	Double	0.418	0.461	1	1.026	0.343	1.536
Actions	*ACTION_USE*	Boolean	True	False	False	True	False	True
	ACTION_UP	Boolean	True	True	True	True	True	True
	ACTION_DOWN	Boolean	True	True	True	True	True	True
	ACTION_RIGHT	Boolean	True	True	True	True	False	True
	ACTION_LEFT	Boolean	True	True	True	True	False	True
Win termination	*winSpriteCounter*	Boolean	True	False	True	False	True	True
	winTimeout	Boolean	False	False	False	True	False	False
	winMultiSpriteCounter	Boolean	False	True	False	False	False	False
	winMultiSpriteCounterSubTypes	Boolean	False	False	False	False	False	False
Lose termination	*loseSpriteCounter*	Boolean	True	False	False	True	True	True
	loseTimeout	Boolean	True	False	False	False	False	False
	loseMultiSpriteCounter	Boolean	False	True	False	False	False	False
	loseSpriteCounterMore	Boolean	False	False	False	False	False	False

- All the features in the group Map.
- All the booleans in the Actions group.

The State Observation object provides lists with the positions of the Immovable, Movable, Portals and Resources sprites, by type. The amounts for each type of Immovable, Movable, Portals and Resources are stored into lists, for each type. As mentioned earlier, the total amount of each is computed at the end in Python: a new column for each type of sprite is added to the data frame with the sum of the amounts. The State Observation object also gives the width and the height of the world, the 2D position of the avatar. The feature *area* is computed by multiplying the width and the height. The *verticalRatio* represents whether the avatar is closer to the top or to the down part of the world map. When this value is closer to zero, it means that the avatar's position is close to the upper side of the map, and when it is closer to one, the avatar's position is close to the bottom edge of the world map. The *horizontalRatio* represents whether the avatar is closer to the left or the right side of the map. Similarly to the *verticalRatio*, when the *horizontalRatio* is closer to zero, the agent is closer to the left edge of the map, while when it is closer to 1, the agent is nearer to the right side. These ratios are computed at the 'data cleaning' step, so after all games have been played by the agent, and two new columns are added in the data frame with this formulae:

$$horizontalRatio = avatarPosition.x/width \qquad (1)$$

$$verticalRatio = avatarPosition.y/height \qquad (2)$$

The booleans in the Actions group are obtained first by storing a list of all the possible actions the player can do when storing the features of the games in the .csv file. Afterwards, a new column for each action is added in the data frame in Python for each of them and a value of 1 or 0 is assigned: 1 for the case the action can be performed by the agent, 0 otherwise.

The features that are obtained from the Game Description object are:

- In the group Total Amount, *nNPC* and *nInteraction*.
- The termination conditions for winning, all the booleans in the Win termination group, and those for winning, all the booleans in the Lose termination group.

The lists of the NPCs and interactions are extracted at the starting of the games, in the .csv files. The *nNPC* and *nInteraction* are obtained by storing the lengths of the lists of NPCs and interactions respectively in the data frame in Python by adding a new column for the *nNPC* and a new column for the *nInteractions*. The Win terminations and Lose terminations are the termination conditions. They are extracted from the Game Description object as lists, and the process to make them as booleans is the same as the one for the actions. The terminations *winSpriteCounter* and *loseSpriteCounter* are conditions to end the game when the amount of a certain sprite, specified in the Game Description, reaches a certain value. For example, a player can win if it reaches a certain amount of sprites that are resources, or it loses when the amount of the resources is below a specified number. The principle is the same for *winMultiSpriteCounter* and *loseMultiSpriteCounter*, but there can be different types of sprites. Regarding *winMultiSpriteCounterSubTypes*, the game ends when the number of sprites of a certain type is equal to a certain value, and the number of different sub-types of the main type is equal to another value. The termination conditions when *winTimeOut* or *loseTimeOut* is true, means that the game ends if the game time reaches a certain value.

The following list, with the corresponding figures, is used to illustrate some features of games:

- The figure 1a shows the level four of the game ”Pokemon” from the framework. In this game there are 8 non-playing characters (NPC) at the beginning of the game. The figure 1b shows the same game but at a later point in time. We can see that some of the NPCs can change types throughout the game.
- ”Sea quest” (level two) is a game that has portals, in the second level there are eight of them, as illustrates figure 1c. Note that for this game,
- ”Pong” is a game where the player can only make vertical moves, so moving the paddle either up or down, it is shown in the Figure 1d.
- In some games, the player can use resources and ”Overload” is one of them. In figure 1e, there are 18 resources, among which one is the sword and 17 are the golden squares.
- A comparison of a game having among the biggest areas, which is ”Cook me pasta”, level two, and the game with the smallest area, ”Painter” level four, are shown in figure 1f and 1g respectively.

2) Agents: As mentioned in Section II-B, there can result many sub-agents by choosing different parameter values. The Move selection strategy, the Playout evaluation and the boolean parameter initBreadthFirst are kept the same for all the 133 agents. The sub-agents used for this thesis are shown in Table XXI. The parameters present in Table XXI of the MaastCTS2 agent are:

- A selection strategy which can either be Progressive History,or OlUct selection strategy, abbreviated respectively as **PH** and **O-L UCT** in Table XXI. The former strategy can take both parameters c, the exploration constant, and w, the constant that determines the influence of progressive bias. The second one is an open-loop UCT selection strategy, that takes only one parameters: c.
- The Playout Strategy can either be the NST playout, the MAST or Random playout. The NST playout is a playout strategy that uses N-Gram selection technique. It takes four parameters: maxPlayoutDepth, epsilon, minNGramVisitCount and maxNGramSize. Two parameters are changed for the playout strategy: maxPlayoutDepth, with the corresponding to the column **Pl1**, and minNGramVisitCount, with the corresponding column **Pl3**. The NST playout takes all the parameters, the MAST take as input the maxPlayoutDepth and epsilon, and the Random playout only takes maxPlayoutDepth.
- **Pl1** is the first parameter of the Playout strategy: the maxPlayoutDepth, which is the maximum depth that the playouts should reach.
- **Pl3** is the parameter minNGramVisitCount that indicates the minimum number of times that an n-gram must have been visited for its statistics to be used.
- The columns **B1**, **B2**, **B3**, **B4**, **B5**, **B6** give the boolean values that the variables noveltyBasedSearch, exploreLosses, knowledgeBasedEval, treeRreuse, alwaysKB and noTreeReuseBFTI take respectively.
- The column **treeR**γ shows the double values that the variable treeReuseGamma have for each agent.
- The values that the parameter maxNumSafetyChecks takes are listed in the column **mNSC**. This parameter defines the maximum number of states that are going to be generated per action.

The victory status of the player, whether the player won, lost or was disqualified, the score and the time step, of game that the sub-agents played are stored in the .csv file in the same row as all the features extracted from that games, as explained in Section III-A1. Each sub-agent played the five levels of the 122 games of the GVGAI Framework five times. In a Python script, all the games played by all the 133 agents are merged into one single data frame. One column for each agent parameter is added to store the value that the parameter takes for the corresponding agent which played the game.

B. Classification methods

Classification methods are used to predict which agent's parameter values would enable it to perform best. The features for the classifiers are the features of the games listed in Section III-A1, and the labels to predict are the parameters of the hyper-agents, detailed in Section III-A2. There is one classifier per label, i.e. per hyper-agent's parameter. There is one model to predict each features independently. The data obtained is sampled to reduce computation time. This is necessary because the final data frame with all games played by agents is composed of 405 650 rows, since there are 133 agents that played the 122 games with 5 levels five times. For this thesis, the predictions are done with Python, using the Scikit-learn machine learning library [20]. The models used are:

- A Dummy Classifier
- A Decision Tree Classifier
- A Random Forest Classifier
- A Multi-layer Perceptron Classifier
- A Logistic Regression Classifier

The Dummy Classifier is used to evaluate how well the more complex classifiers are to predict the values. The scores of this classifier is compared to the models' best scores after parameter tuning, and the models with their parameters having default values. To get the scores of the models with default parameter values, 10-fold validation is performed. Regarding hyper-parameter tuning, the ‘GridSearchCV‘ method, from the scikit-learn library [20], is used to tune the specified parameters for each type of classifier, perform a 5-fold cross-validation, and fit the model with the highest score on the training data.

- Decision Tree classifier: two hyper-parameters for this classifier are tuned: max_depth and max_leaf_nodes. The parameter max_depth is tuned to take the values in the interval from 1 to 13 included, with the space between the elements being of 3: 1, 4, 7, 10, 13. The max_leaf_nodes parameter takes the values between 5 and 100, evenly spaced by 20.

- Random Forest classifier: three hyper-parameters are tuned: max_depth, max_leaf_nodes and $n\_estimators$. The two former are tuned in the same way as for the Decision Tree Classifier, and the $n\_estimators$ parameter take the values between 1 and 21 included, evenly spaced by 5.
- Multi-layer Perceptron classifier: the only parameter adjusted for this classifier is max_iter. It can take as values the tuples (100) (one hidden layer), (50,50,50) (three hidden layers), (50,100,50) (three hidden layers), (50, 50, 50, 50) (four hidden layers).
- Logistic Regression classifier: the two adjusted parameters for this classifier are the $solver$ and C. The $solver$ might either take as value sag, $saga$ or $lbfgs$, and C is tuned to take the values [1, 5, 10].

C. Agents' performance

The sklearn-porter library [21] is used to transpile trained classifiers in sklearn into other programming languages: in Java, C, JavaScript, Go, PHP and Ruby. Since for this thesis the GVGAI framework (written in Java) is used, the estimators are transpiled into Java classes. After performing hyper-parameter tuning for each agent's parameter, the classifier with the highest score for that parameter is transpiled into a Java class file. The performance of the agent with parameters chosen for the competition of 2014, and the agents which parameters are chosen by the estimators, are compared by making them play 20 games, listed in Table III. Among the 20 games, 10 games where the number of lost ones is the biggest (and where the level chosen is 0) and 10 randomly chosen games, with randomly chosen levels. Table III also shows in parentheses the number of game that resulted in a loss.

TABLE III: Games to test performance

Randomly chosen	**With the most losses (number of lost games across all levels)**
bomber	assemblyline (3314)
camelRace	bird (3325)
chopper	brainman (3317)
lasers	chainreaction (3325)
mario	digdug (3325)
overload	fireman (3325)
pong	lasers2 (3324)
run	lemmings (3325)
surround	vortex (3318)
thecitadel	witnessprotected (3325)

IV. Results

A. Classification models - hyper-parameter tuning

Table IV shows the mean of the predictions' scores for each model with default value parameters, and the Dummy Classifier. Tables V, VI, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, XVI, XVII and XVIII show for each agent's parameter the the parameters enable the classification methods to have higher accuracy, as well as the maximum accuracy score for that parameter.

TABLE V: Best models for Selection Strategy

Model type	**Best parameters**	**score**
DT	$max\_depth = 13$, $max\_leaf\_nodes = 85$	0.532444
RF	$max\_depth = 7$, $max\_leaf\_nodes = 85$, $n\_estimators = 11$	0.533852
LG	$C = 1$, $solver = sag$	0.515928
MLP	$hidden\_layer\_size = (50, 100, 50)$	0.516368

TABLE VI: Best models for parameter c

Model type	**Best parameters**	**score**
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.977911
RF	$max\_depth = 1$, $max\_leaf\_nodes = 5$, $n\_estimators = 1$	0.977911
LG	$C = 1$, $solver = sag$	0.977911
MLP	$hidden\_layer\_size = (100)$	0.977911

TABLE VII: Best models for parameter w

Model type	**Best parameters**	**score**
DT	$max\_depth = 10$, $max\_leaf\_nodes = 85$	0.527779
RF	$max\_depth = 13$, $max\_leaf\_nodes = 25$, $n\_estimators = 11$	0.529129
LG	$C = 1$, $solver = sag$	0.508302
MLP	$hidden\_layer\_size = (50, 50, 50, 50)$	0.508302

TABLE VIII: Best models for Playout Strategy

Model type	**Best parameters**	**score**
DT	$max\_depth = 7$, $max\_leaf\_nodes = 25$	0.339924
RF	$max\_depth = 4$, $max\_leaf\_nodes = 45$, $n\_estimators = 11$	0.343678
LG	$C = 5$, $solver = sag$	0.340569
MLP	$hidden\_layer\_size = (50, 50, 50, 50)$	0.338457

TABLE IX: Best models for parameter maxPlayoutDepth

Model type	**Best parameters**	**score**
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.539132
RF	$max\_depth = 4$, $max\_leaf\_nodes = 65$, $n\_estimators = 16$	0.539689
LG	$C = 1$, $solver = sag$	0.539132
MLP	$hidden\_layer\_size = (50, 100, 50)$	0.508506

TABLE IV: Accuracy of the classifier models without hyper-parameter tuning

	Dummy Classifier	**Decision Tree**	**Random Forest**	**Multi-layer Perceptron**	**Logistic regression**
Labels					
Selection Strategy	0.5889	0.5889	0.5901	0.55432	0.5889
c	0.9244	0.9241	0.92435	0.92438	0.9244
w	0.5512	0.5445	0.54502	0.5512	0.5512
Playout Strategy	0.5484	0.5455	0.5458	0.39538	0.5484
maxPlayoutDepth	0.9269	0.9268	0.9268	0.9268	0.9269
minNGramVisitCount	0.51	0.4375	0.45435	0.49850	0.5081
noveltyBasedPrunning	0.9612	0.961	0.9611	0.7767	0.9612
exploreLosses	0.7689	0.7698	0.7697	0.66132	0.7689
knowledgeBasedEval	0.9338	0.933	0.9333	0.7603	0.9338
treeReuse	0.9609	0.9605	0.9607	0.9609	0.9609
treerReuseGamma	0.9589	0.9588	0.9588	0.9589	0.9589
maxNumSafetyChecks	0.9196	0.9195	0.9195	0.743	0.9196
alwaysKB	0.8833	0.883	0.8832	0.8802	0.8833
noTreeReuseBFTI	0.961	0.9609	0.9609	0.961	0.961

TABLE X: Best models for parameter minNGramVisitCount

Model type	**Best parameters**	**score**
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.662628
RF	$max\_depth = 4$, $max\_leaf\_nodes = 65$, $n\_estimators = 1$	0.662716
LG	$C = 1$, $solver = sag$	0.662628
MLP	$hidden\_layer\_size = (50, 50, 50)$	0.662628

TABLE XI: Best models for boolean noveltyBasedPrunning

Model type	**Best parameters**	**score**
DT	$max\_depth = 4$, $max\_leaf\_nodes = 5$	0.914432
RF	$max\_depth = 10$, $max\_leaf\_nodes = 45$, $n\_estimators = 11$	0.914432
LG	$C = 1$, $solver = sag$	0.914403
MLP	$hidden\_layer\_size = (50, 50, 50)$	0.914403

TABLE XII: Best models for boolean exploreLosses

Model type	**Best parameters**	**score**
DT	$max\_depth = 13$, $max\_leaf\_nodes = 25$	0.883250
RF	$max\_depth = 13$, $max\_leaf\_nodes = 85$, $n\_estimators = 16$	0.883837
LG	$C = 1$, $solver = sag$	0.882957
MLP	$hidden\_layer\_size = (50, 50, 50, 50)$	0.882957

TABLE XIII: Best models for boolean knowledgeBasedEval

Model type	**Best parameters**	**score**
DT	$max\_depth = 7$, $max\_leaf\_nodes = 25$	0.920534
RF	$max\_depth = 13$, $max\_leaf\_nodes = 85$, $n\_estimators = 11$	0.920563
LG	$C = 1$, $solver = sag$	0.920211
MLP	$hidden\_layer\_size = (50, 100, 50)$	0.920211

TABLE XIV: Best models for boolean treeReuse

Model type	**Best parameters**	**score**
DT	$max\_depth = 10$, $max\_leaf\_nodes = 45$	0.913376
RF	$max\_depth = 13$, $max\_leaf\_nodes = 65$, $n\_estimators = 16$	0.913376
LG	$C = 1$, $solver = sag$	0.912848
MLP	$hidden\_layer\_size = (50, 100, 50)$	0.912848

TABLE XV: Best models for boolean treeReuseGamma

Model type	**Best parameters**	**score**
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.820974
RF	$max\_depth = 1$, $max\_leaf\_nodes = 5$, $n\_estimators = 1$	0.820974
LG	$C = 1$, $solver = sag$	0.820974
MLP	$hidden\_layer\_size = (50, 50, 50)$	0.820974

TABLE XVI: Best models for boolean maxNumSafetyChecks

Model type	**Best parameters**	**score**
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.813963
RF	$max\_depth = 1$, $max\_leaf\_nodes = 5$, $n\_estimators = 1$	0.813963
LG	$C = 1$, $solver = sag$	0.813963
MLP	$hidden\_layer\_size = (50, 100, 50)$	0.813963

TABLE XVII: Best models for boolean alwaysKB

Model type	**Best parameters**	**score**
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.90176
RF	$max\_depth = 1$, $max\_leaf\_nodes = 5$, $n\_estimators = 6$	0.90176
LG	$C = 1$, $solver = sag$	0.90176
MLP	$hidden\_layer\_size = (50, 50, 50)$	0.90176

TABLE XVIII: Best models for boolean noTreeReuseBFTI

Model type	Best parameters	score
DT	$max\_depth = 1$, $max\_leaf\_nodes = 5$	0.908976
RF	$max\_depth = 1$, $max\_leaf\_nodes = 5$, $n\_estimators = 1$	0.908976
LG	$C = 1$, $solver = sag$	0.908976
MLP	$hidden\_layer\_size = (100)$	0.908976

B. Agents' performance

Tables XIX and XX shows the the performance of the MaastCTS2 agent with competition parameters, and of the agents which parameters where chosen by the trained estimator. The victory status, the score, the time step are listed, as well as the parameter(s) that are predicted to be different from the MaastCTS2 agent used for the competition. Table XIX shows the performances for the games that had the most losses, and XX shows the performances of the agents for randomly selected games. The victory status for one game can be 1 for a win, 0 for a loss, -100 when the agent is disqualified. In both tables, the victory status is the sum of these results of the games that were played five times. The scores are the mean of the scores for that game played five times. It can be observed in Table XIX that the victory status and the scores did not change in most cases. For the games in Table XX, it can be noticed that the agent was "improved" by the classification models in pong, mario, lasers (it didn't win, any game but it was not disqualified). The agent with competition parameters did better in the game thecitadel. One observation that can be made is that the change is score corresponds to the change in the victory status, so if an agent won more games, it has a higher score as in the games thecitadel and lasers for instance. The table also shows that in the games where the "improved" agents won, or performed as well as the initial one, the time steps are lower, not significantly however (except for the game pong).

V. Discussion

A. Classification models - hyper-parameter tuning

In Table IV, several observations can be made:

- For any parameter, the differences in scores is rather low except for the parameters
- The results of the Dummy Classifierand the Logistic Regression classifiers are the same when predicting any parameter.
- The Decision Tree classifier is the one that predicted better than the others only once.
- The Random Forest Classifier is the second could predict better than the other models for two parameters.
- The MLP predicted better than the Decision Trees and the Random Forest Classifiers, but not more than the Dummy Classifier and the Logistic Regression Classifier. In some cases, the accuracy scores for this classifier are significantly lower than the other classifier. For instance, the accuracy for predicting maxNumSafetyChecks is around 0.17 lower than the accuracy of other models. Similar observation can be made for the parameter knowledgeBasedEval, exploreLosses, noveltyBasedEval, and the Playout Strategy. We can conclude that the MLP classifier without parameter tuning is not the most accurate for these features, especially the boolean variables.

Regarding hyper-parameter tuning, models for some of the agent's parameters didn't output more accurate results, some even did worse: the models for the Selection Strategy, the parameter w, the Playout Strategy, maxPlayoutDepth (which difference is very big, the scores of the models with tuned parameter are around 0.4 points lower than the model with default parameters), noveltyBasedPrunning, knowledgeBasedEval (except for the MLP classification model), treeReuse, treeReuseGamma, maxNumSafetyChecks and noTreeReuseBFTI. The parameters where the models with adjusted parameters output more accurate results are the parameter c, minNGramVisitCount, exploreLosses, and alwaysKB. For the majority of the hyper-agent's parameters, the models which parameters were tuned performed worse at predicting the right parameter values than the ones which parameters were tuned. One hypothesis to explain this is that the tuned parameters did not have the right values to be tuned, or that other parameters of this models might have been useful to tune to increase their accuracy.

B. Agents' performance

The performance, the victory status, of the agents with parameters adjusted by the models is not improved compared the initial agent for the most lost games. However, from the observations made in Section IV-B, the agents results improved slightly in more games, the ones randomly chosen, than they got worse. The difference in performance can be said significant in games mario, lasers and pong, as discussed in Section IV-B. Also, the average time steps needed to win games also decreased, which can also be considered as an improvement to the competition agent.

VI. Conclusion

The accuracy of the results did not improve after parameter tuning, which might be cause by the fact that the parameters tuned did not take the most ¿¿¿ values, or that some other parameters of the models should have been tuned. To answer the research question 1, the accuracy results of the models without tuned parameter were better that the one of the MLP classifier, and the four models with the specified parameter values. For this thesis, classification methods were used to improve the agent's performance by specifying the values from the trained models. These models were trained in Python using the Scikit-learn library, and transpiled into Java classes that were used to choose the values for the parameters of the agents individually, this answers the research question 2. Regarding research question 3, the classification methods enabled the agents to win more games than the agent, and to win some

TABLE XIX: Performance of agents - games with higher number of lost

Game	Victory		Scores mean		Time step		Parameters chosen by the model
	Competition	Predicted	Competition	Predicted	Competition	Predicted	
assemblyline	0	0	6.6	6.4	1500.0	1500.0	RandomPlayout(10.0)
bird	0	0	0.0	0.0	141.0	141.0	O-L UCT(0.6), RandomPlayout(10.0)
brainman	0	0	11.0	11.0	2000.0	2000.0	RandomPlayout(10.0)
chainreaction	0	0	4.8	4.8	1500.0	1500.0	RandomPlayout(10.0)
digdug	0	0	15.4	6.0	1057.8	381.4	O-L UCT(0.6), RandomPlayout(10.0)
fireman	0	0	-11.4	-1.2	1371.8	1265.0	RandomPlayout(10.0)
lasers2	0	0	0.0	0.0	1834.0	888.0	RandomPlayout(10.0)
lemmings	0	0	-0.8	0.0	2000.0	2000.0	RandomPlayout(10.0)
vortex	0	0	1.0	1.0	1000.0	1000.0	RandomPlayout(10.0)
witnessprotected	0	0	0.0	0.0	511.8	496.2	RandomPlayout(10.0)

TABLE XX: Performance of agents - randomly chosen games

Game	Victory		Scores mean		Time step		Parameters chosen by the model
	Competition	Predicted	Competition	Predicted	Competition	Predicted	
bomber	2	2	3.2	3.2	837.4	798.8	RandomPlayout(10.0)
camelRace	5	5	1.0	1.0	67.0	57.8	RandomPlayout(10.0)
chopper	5	5	18.8	19.6	880.8	912.0	RandomPlayout(10.0)
lasers	-200	0	-400.0	0.0	714.0	1574.8	RandomPlayout(10.0)
mario	-300	1	-598.2	5.6	783.2	536.6	RandomPlayout(10.0)
overload	5	5	17.2	17.0	241.2	211.8	RandomPlayout(10.0)
pong	-197	5	-399.4	1.0	388.8	853.2	O-L UCT(0.6)
run	0	0	0.0	0.0	132.6	123.8	RandomPlayout(10.0)
surround	5	5	1.0	1.0	0.0	0.0	RandomPlayout(10.0)
thecitadel	5	3	6.0	4.4	13.8	864.8	RandomPlayout(10.0)

of them faster. Therefore, classification models improved the game playing of the hyper-agent.

VII. Future work

To test further the accuracy of the models, different approaches could be done:

- Use other features of the games, like the types of the other avatars, whether they are shooting etc which can be extended thanks to the information stored by the class StoreToCSV.java.
- In this thesis, the parameters of the agents, the labels of the models, were not predicted together, but independently. An improvement would be to have more than one, in the best case all, the parameters predicted to make one sub-agent that can play the game with some specific features.
- Some game generation has been done in the past, among other also for the competition, where the participants, on top of developing playing agents, had to submit good game and rule generators. To test the accuracy of the model, new games with new rules could be generated, and the agent would be able to play more games and its performance with the classification method included would be tested.

References

[1] P. K. Y. Yap, N. Burch, R. C. Holte, and J. Schaeffer, “Any-angle path planning for computer games,” in *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[2] J. Lisowski, “Dynamic games methods in navigator decision support system for safety navigation,” in *Proceedings of the European safety and reliability conference*, vol. 2, pp. 1285–1292, 2005.

[3] 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.

[4] X. Cui and H. Shi, “A*-based pathfinding in modern computer games,” *International Journal of Computer Science and Network Security*, vol. 11, no. 1, pp. 125–130, 2011.

[5] J. Schaeffer and H. J. Van den Herik, “Games, computers, and artificial intelligence,” *Artificial Intelligence*, vol. 134, no. 1-2, pp. 1–7, 2002.

[6] M. Genesereth, N. Love, and B. Pell, “General game playing: Overview of the aaai competition,” *AI magazine*, vol. 26, no. 2, pp. 62–62, 2005.

[7] R. Tagiew, “General game management agent,” *arXiv preprint arXiv:0903.0353*, 2009.

[8] G. Fong, “Adapting cots games for military simulation,” in *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pp. 269–272, 2004.

[9] N. Cummins, “Integrating e-commerce and games,” *Personal and Ubiquitous Computing*, vol. 6, no. 5, pp. 362–370, 2002.

[10] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, “General video game playing,” *Artificial and Computational Intelligence in Games*, pp. 76–83, 2013.

[11] S. Risi and J. Togelius, “Increasing generality in machine learning through procedural content generation,” *Nature Machine Intelligence*, vol. 2, no. 8, pp. 428–436, 2020.

[12] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A neuroevolution approach to general atari game playing,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014.

[13] C. Crawford, *Chris Crawford on interactive storytelling*. New Riders, 2012.

[14] A. Mendes, J. Togelius, and A. Nealen, “Hyper-heuristic general video

game playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 94–101, IEEE, 2016.

[15] D. J. Soemers, C. F. Sironi, T. Schuster, and M. H. Winands, "Enhancements for real-time monte-carlo tree search in general video game playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 436–443, IEEE, 2016.

[16] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[17] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," *Artificial and Computational Intelligence in Games*, p. 85, 2013.

[18] "Single player plannning championship 2016." http://gvgai.net/championship.php?t=2016&t=sp.

[19] R. D. Gaina, A. Couëtoux, D. J. Soemers, M. H. Winands, T. Vodopivec, F. Kirchgeßner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 two-player gvgai competition," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, 2017.

[20] "scikit-learn: machine learning in python — scikit-learn 1.0.1 documentation." https://scikit-learn.org/stable/index.html.

[21] D. Morawiec, "sklearn-porter." Transpile trained scikit-learn estimators to C, Java, JavaScript and others.

Appendix

TABLE XXI: Sub-agents with different parameter values

N^o	Selection	c	w	Playout	Pl1	Pl3	B1	B2	B3	B4	treeRγ	mNSC	B5	B6
1	PH	0.6	1.0	NST	10.0	7.0	true	true	true	true	0.6	3	true	false
2	PH	0.0	1.0	NST	10.0	7.0	true	true	true	true	0.6	3	true	false
3	PH	0.9	1.0	NST	10.0	7.0	true	true	true	true	0.6	3	true	false
4	PH	0.6	0.5	NST	10.0	7.0	true	true	true	true	0.6	3	true	false
5	PH	0.6	1.0	MAST	10.0	null	true	true	true	true	0.6	3	true	false
6	PH	0.6	1.0	Random	10.0	null	true	true	true	true	0.6	3	true	false
7	PH	0.6	1.0	NST	10.0	9.0	true	true	true	true	0.6	3	true	false
8	PH	0.6	1.0	NST	10.0	7.0	false	true	true	true	0.6	3	true	false
9	PH	0.6	1.0	NST	10.0	7.0	true	false	true	true	0.6	3	true	false
10	PH	0.6	1.0	NST	10.0	7.0	true	true	false	true	0.6	3	true	false
11	PH	0.6	1.0	NST	10.0	7.0	true	true	true	false	0.6	3	true	false
12	PH	0.6	1.0	NST	10.0	7.0	true	true	true	true	0.3	3	true	false
13	PH	0.6	1.0	NST	10.0	7.0	true	true	true	true	0.6	5	true	false
14	PH	0.6	1.0	NST	10.0	7.0	true	true	true	true	0.6	9	true	false
15	PH	0.6	1.0	NST	10.0	7.0	true	true	true	true	0.6	3	false	false
16	PH	0.6	1.0	NST	10.0	7.0	true	true	true	true	0.6	3	true	true
17	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.6	3	true	false
18	O-L UCT	0.6	null	Random	10.0	null	true	true	true	true	0.6	3	true	false
19	O-L UCT	0.6	null	Random	6.0	null	true	true	true	true	0.6	3	true	false
20	O-L UCT	0.6	null	Random	6.0	null	true	false	true	true	0.6	3	true	false
21	O-L UCT	0.9	null	Random	10.0	null	true	false	true	true	0.6	3	true	false
22	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.6	3	true	false
23	O-L UCT	0.6	null	MAST	10.0	null	true	false	true	true	0.6	3	false	false
24	PH	0.6	0.1	MAST	10.0	null	false	true	true	true	0.6	3	true	false
25	PH	0.6	0.1	MAST	10.0	null	true	false	true	true	0.6	3	true	false
26	PH	0.6	0.1	MAST	10.0	null	true	true	false	true	0.6	3	true	false
27	PH	0.6	0.1	MAST	10.0	null	true	true	true	false	0.6	3	true	false
28	PH	0.6	0.1	MAST	10.0	null	true	true	true	true	0.6	3	false	false
29	PH	0.6	0.1	MAST	10.0	null	true	true	true	true	0.6	3	true	true
30	PH	0.6	0.1	MAST	10.0	null	true	true	true	true	0.6	5	true	false
31	PH	0.6	0.1	MAST	10.0	null	true	true	true	true	0.6	9	true	false
32	PH	0.6	0.1	MAST	10.0	null	true	true	true	true	0.3	3	true	false
33	PH	0.6	0.1	MAST	10.0	null	true	true	true	true	0.8	3	true	false
34	PH	0.6	0.1	MAST	7.0	null	false	true	true	true	0.6	3	true	false
35	PH	0.6	0.1	MAST	7.0	null	true	false	true	true	0.6	3	true	false
36	PH	0.6	0.1	MAST	7.0	null	true	true	false	true	0.6	3	true	false
37	PH	0.6	0.1	MAST	7.0	null	true	true	true	false	0.6	3	true	false
38	PH	0.6	0.1	MAST	7.0	null	true	true	true	true	0.6	3	false	false
39	PH	0.6	0.1	MAST	7.0	null	true	true	true	true	0.6	3	true	true
40	PH	0.6	0.1	MAST	7.0	null	true	true	true	true	0.6	5	true	false
41	PH	0.6	0.1	MAST	7.0	null	true	true	true	true	0.6	9	true	false
42	PH	0.6	0.1	MAST	7.0	null	true	true	true	true	0.3	3	true	false
43	PH	0.6	0.1	MAST	7.0	null	true	true	true	true	0.8	3	true	false
44	O-L UCT	0.6	null	MAST	7.0	null	false	true	true	true	0.6	3	true	false
45	O-L UCT	0.6	null	MAST	7.0	null	true	false	true	true	0.6	3	true	false
46	O-L UCT	0.6	null	MAST	7.0	null	true	true	false	true	0.6	3	true	false
47	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	false	0.6	3	true	false
48	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	true	0.6	3	false	false

Continued on next page

TABLE XXI – continued from previous page

N^o	Selection	c	w	Playout	Pl1	Pl4	B1	B2	B3	B4	treeRγ	mNSC	B5	B6
49	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	true	0.6	3	true	true
50	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	true	0.6	5	true	false
51	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	true	0.6	9	true	false
52	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	true	0.3	3	true	false
53	O-L UCT	0.6	null	MAST	7.0	null	true	true	true	true	0.8	3	true	false
54	O-L UCT	0.6	null	MAST	10.0	null	false	true	true	true	0.6	3	true	false
55	O-L UCT	0.6	null	MAST	10.0	null	true	false	true	true	0.6	3	true	false
56	O-L UCT	0.6	null	MAST	10.0	null	true	true	false	true	0.6	3	true	false
57	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	false	0.6	3	true	false
58	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.6	3	false	false
59	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.6	3	true	true
60	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.6	5	true	false
61	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.6	9	true	false
62	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.3	3	true	false
63	O-L UCT	0.6	null	MAST	10.0	null	true	true	true	true	0.8	3	true	false
64	O-L UCT	0.6	null	NST	10.0	7.0	false	true	true	true	0.6	3	true	false
65	O-L UCT	0.6	null	NST	10.0	7.0	true	false	true	true	0.6	3	true	false
66	O-L UCT	0.6	null	NST	10.0	7.0	true	true	false	true	0.6	3	true	false
67	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	false	0.6	3	true	false
68	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.6	3	false	false
69	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.6	3	true	true
70	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.6	5	true	false
71	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.6	9	true	false
72	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.3	3	true	false
73	O-L UCT	0.6	null	NST	10.0	7.0	true	true	true	true	0.8	3	true	false
74	O-L UCT	0.6	null	NST	7	7.0	false	true	true	true	0.6	3	true	false
75	O-L UCT	0.6	null	NST	7	7.0	true	false	true	true	0.6	3	true	false
76	O-L UCT	0.6	null	NST	7	7.0	true	true	false	true	0.6	3	true	false
77	O-L UCT	0.6	null	NST	7	7.0	true	true	true	false	0.6	3	true	false
78	O-L UCT	0.6	null	NST	7	7.0	true	true	true	true	0.6	3	false	false
79	O-L UCT	0.6	null	NST	7	7.0	true	true	true	true	0.6	3	true	true
80	O-L UCT	0.6	null	NST	7	7.0	true	true	true	true	0.6	5	true	false
81	O-L UCT	0.6	null	NST	7	7.0	true	true	true	true	0.6	9	true	false
82	O-L UCT	0.6	null	NST	7	7.0	true	true	true	true	0.3	3	true	false
83	O-L UCT	0.6	null	NST	7	7.0	true	true	true	true	0.8	3	true	false
84	PH	0.6	1.0	NST	7	7.0	false	true	true	true	0.6	3	true	false
85	PH	0.6	1.0	NST	7	7.0	true	false	true	true	0.6	3	true	false
86	PH	0.6	1.0	NST	7	7.0	true	true	false	true	0.6	3	true	false
87	PH	0.6	1.0	NST	7	7.0	true	true	true	false	0.6	3	true	false
88	PH	0.6	1.0	NST	7	7.0	true	true	true	true	0.6	3	false	false
89	PH	0.6	1.0	NST	7	7.0	true	true	true	true	0.6	3	true	true
90	PH	0.6	1.0	NST	7	7.0	true	true	true	true	0.6	5	true	false
91	PH	0.6	1.0	NST	7	7.0	true	true	true	true	0.6	9	true	false
92	PH	0.6	1.0	NST	7	7.0	true	true	true	true	0.3	3	true	false
93	PH	0.6	1.0	NST	7	7.0	true	true	true	true	0.8	3	true	false
94	PH	0.6	1.0	Random	10	null	false	true	true	true	0.6	3	true	false
95	PH	0.6	1.0	Random	10	null	true	false	true	true	0.6	3	true	false
96	PH	0.6	1.0	Random	10	null	true	true	false	true	0.6	3	true	false
97	PH	0.6	1.0	Random	10	null	true	true	true	false	0.6	3	true	false
98	PH	0.6	1.0	Random	10	null	true	true	true	true	0.6	3	false	false
99	PH	0.6	1.0	Random	10	null	true	true	true	true	0.6	3	true	true

Continued on next page

TABLE XXI – continued from previous page

N^o	Selection	c	w	Playout	Pl1	Pl4	B1	B2	B3	B4	treeRγ	mNSC	B5	B6
100	PH	0.6	1.0	Random	10	null	true	true	true	true	0.6	5	true	false
101	PH	0.6	1.0	Random	10	null	true	true	true	true	0.6	9	true	false
102	PH	0.6	1.0	Random	10	null	true	true	true	true	0.3	3	true	false
103	PH	0.6	1.0	Random	10	null	true	true	true	true	0.8	3	true	false
104	PH	0.6	1.0	Random	7	null	false	true	true	true	0.6	3	true	false
105	PH	0.6	1.0	Random	7	null	true	false	true	true	0.6	3	true	false
106	PH	0.6	1.0	Random	7	null	true	true	false	true	0.6	3	true	false
107	PH	0.6	1.0	Random	7	null	true	true	true	false	0.6	3	true	false
108	PH	0.6	1.0	Random	7	null	true	true	true	true	0.6	3	false	false
109	PH	0.6	1.0	Random	7	null	true	true	true	true	0.6	3	true	true
110	PH	0.6	1.0	Random	7	null	true	true	true	true	0.6	5	true	false
111	PH	0.6	1.0	Random	7	null	true	true	true	true	0.6	9	true	false
112	PH	0.6	1.0	Random	7	null	true	true	true	true	0.3	3	true	false
113	PH	0.6	1.0	Random	7	null	true	true	true	true	0.8	3	true	false
114	O-L UCT	0.6	null	Random	7	null	false	true	true	true	0.6	3	true	false
115	O-L UCT	0.6	null	Random	7	null	true	false	true	true	0.6	3	true	false
116	O-L UCT	0.6	null	Random	7	null	true	true	false	true	0.6	3	true	false
117	O-L UCT	0.6	null	Random	7	null	true	true	true	false	0.6	3	true	false
118	O-L UCT	0.6	null	Random	7	null	true	true	true	true	0.6	3	false	false
119	O-L UCT	0.6	null	Random	7	null	true	true	true	true	0.6	3	true	true
120	O-L UCT	0.6	null	Random	7	null	true	true	true	true	0.6	5	true	false
121	O-L UCT	0.6	null	Random	7	null	true	true	true	true	0.6	9	true	false
122	O-L UCT	0.6	null	Random	7	null	true	true	true	true	0.3	3	true	false
123	O-L UCT	0.6	null	Random	7	null	true	true	true	true	0.8	3	true	false
124	O-L UCT	0.6	null	Random	10	null	false	true	true	true	0.6	3	true	false
125	O-L UCT	0.6	null	Random	10	null	true	false	true	true	0.6	3	true	false
126	O-L UCT	0.6	null	Random	10	null	true	true	false	true	0.6	3	true	false
127	O-L UCT	0.6	null	Random	10	null	true	true	true	false	0.6	3	true	false
128	O-LUCT	0.6	null	Random	10	null	true	true	true	true	0.6	3	false	false
129	O-L UCT	0.6	null	Random	10	null	true	true	true	true	0.6	3	true	true
130	O-L UCT	0.6	null	Random	10	null	true	true	true	true	0.6	5	true	false
131	O-L UCT	0.6	null	Random	10	null	true	true	true	true	0.6	9	true	false
132	O-L UCT	0.6	null	Random	10	null	true	true	true	true	0.3	3	true	false
133	O-L UCT	0.6	null	Random	10	null	true	true	true	true	0.8	3	true	false