University of Maribor

Faculty of Electrical Engineering
and Computer Science

FERI

Game Feature Validation of a Real-Time Game Space with an eXtended Classifier System

25th IEEE International Conference on Intelligent Engineering Systems 2021
(INES 2021)

Authors:
Damijan Novak (damijan.novak@um.si) and Iztok Fister Jr. (iztok.fister1@um.si)

7.7.2021

PRESENTATION AGENDA

1. Motivation and purpose of the article
2. Game feature
3. Game space (of Tic-Tac-Toe)
4. eXtended Classifier System (XCS)
   a) Description
   b) Syntactic and Semantic Validation of Classifiers
5. From game playing to play-testing (Validation component)
6. Proof-of-concept / Experiment
7. Future work

MOTIVATION

- Development of the new game space is a challenging task.

game space > game world > game feature > game mechanics

- Testing is hard due to the complexity of game spaces.

- It requires lots of time and (human) resources to create and test a game space.

- Automatic game testing is a largely untouched niche, and it still relies mainly on manual testing.

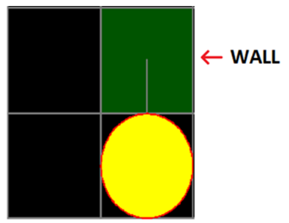
PURPOSE

- We tackle the time-consuming game space validation problem with a “proof-of-concept" utilization of automated validation of game features of a Tic-tac-toe game space through the usage of an eXtended Classifier System (XCS) algorithm.

- We also focus on obtaining results in real-time. By real-time we are not referring to hard, but to the soft real-time (games can be considered as soft real-time applications).

GAME FEATURE

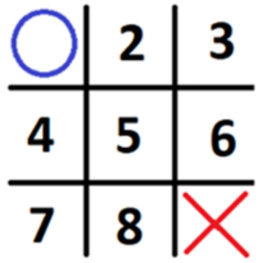
- Game features: a generic term used to refer to differences and similarities between games, which is further refined by the terms ”game elements” and ”game attributes”
- Game mechanics: are methods invoked by agents for interacting with the game world.
  - Game mechanics connects with each other the game rules, the game objects and their properties, and the game environment, so that immersive game scenarios are created where a player can progress through the game in a predictive way.
- Game feature becomes the holder, or reflects the specific characteristic of the game (e.g. A game object wall for which the game mechanics defined indestructibility becomes the important game feature placed in the story of a prison break).



GAME SPACE (OF TIC-TAC-TOE)

- Full search space of the game space (also known as state-space): is defined as a set S, which holds all the states of all possible configurations of the game map.

- Full search space of Tic-Tac-Toe holds 3^9 (19.683) states.

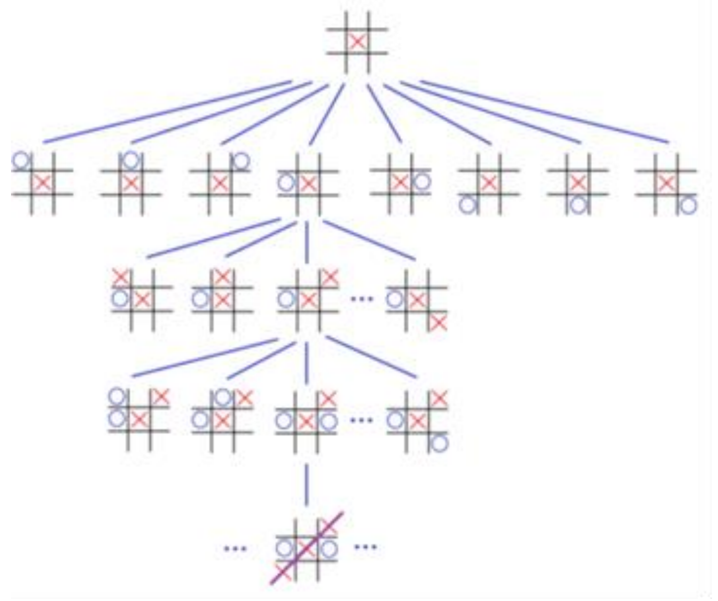


- The legal (reduced) game search space can be defined as set S, which contains only the legal layouts of the game board that can be achieved from the starting position of the game.

- From the whole state-space of the game Tic-Tac-Toe (19.683), we are left with 5.478 legal states, after the illegal ones have been excluded (e.g. six crosses present on the game board, but no circles).

„PROOF-OF-CONCEPT"

- Full (and many times even reduced) search spaces can be overwhelming from the available computer resources standing.
- Low number of legal games and useful states of Tic-Tac-Toe is very useful game space example for „Proof-of-concept" research (such as we are trying to achieve with this article of game feature validation by usage of XCS).



LEARNING CLASSIFIER SYSTEMS (LCS)

- The XCS algorithms belong to the group of Learning Classifier Systems (LCS), which are rule-based systems.
- Rules are kept in a population set, which (as a whole set) represents the candidate solution to the problem.
- Rule / Classifier
  - IF condition THEN action
- Condition is a vector, and it represents one form of the environment, while the action part stands for the action that the classifier is propagating. The vector of the condition can be a mix of bits (0 and 1) and #'s (don't care, or not relevant bits).
  - # symbol is introduced with the purpose of conditions being able to generalize.

EXTENDED CLASSIFIER SYSTEM (XCS)

- LCS's learn and evolve new classifiers through interaction with the environment.
  - By executing actions for which the indication of the value of the action, called reward, is received.
- We chose the XCS algorithm, because the literature shows that it can learn a state-value function over the complete state-action space (*under certain criteria) through efficient generalizations.
  - Important part of LCS is also a component of the genetic algorithm.
- The XCS classifiers also hold additional parameters, which contribute importantly to the XCS's efficient generalizations:
  - Prediction estimate (payoff that we can expect if the condition of the classifier will match the environment input),
  - Prediction error (estimates the errors made in the prediction)
  - Experience (a counter of how many times the classifier has belonged to the set of chosen (propagating) actions).
  - Fitness tells us how accurate the prediction estimate is, since we are not just interested in the prediction estimate, but also in how accurate this prediction is.

SYNTACTIC AND SEMANTIC VALIDATION OF CLASSIFIERS

- To connect the XCS algorithm successfully to the game space of Tic-Tac-Toe, we need to have the syntactically and semantically valid classifiers.

- The validation is needed, because otherwise an XCS cycle could get stuck, or wouldn’t generalize efficiently.

- The invalidness of classifiers can occur as a result of the random classifier creation (e.g. random creation of the population set), while executing classifier covering (which can occur during match set creation), or during the execution of the genetic algorithm (crossover operations).

SYNTACTIC AND SEMANTIC VALIDATION OF CLASSIFIERS

- With the syntactic validation we ensure that the condition of the classifier (represented as one variation of the game state), and the action, which will be executed upon the game state, are compatible.

- With semantic validation we make sure if the chosen action of an XCS algorithm can be executed on the live game state (e.g. we test if we are putting a symbol on the empty cell).

- The semantic validation is not performed on the condition, but always on the live game state (the state of the environment).

- If the classifier during either validation doesn’t check out, such a classifier is disregarded for future use.

FROM GAME PLAYING TO PLAY-TESTING

- Game agents are vital parts of many games, with the main assumption of their function being intelligent and rational.

- Intelligent game agents are designed to be good performers while playing the game, so we would like to use them for game feature validation as well.

- To validate game features with a game agent, we need to adapt its game-playing mechanics for the play-testing purpose.

VALIDATION COMPONENT

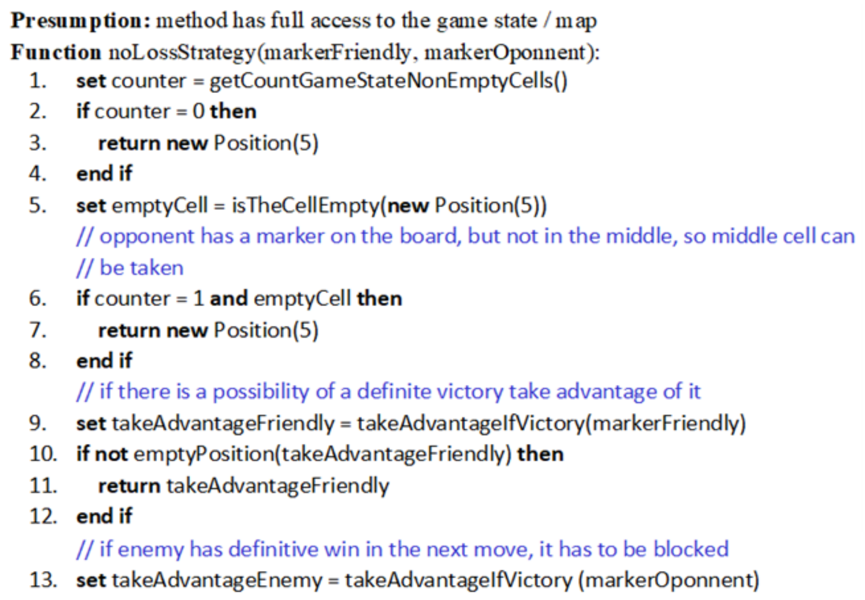
- In our case, we created and used the game constituent part we call a (non-invasive) validation component.
- The non-invasive component is summed up in the following steps:

1. Acquiring the values of a measure: the component needs the measured data about the agent`s workings (actions that were performed) and information about the game state.
2. Executing of the internal method: designed to validate the specific game feature. The method is the center piece and crucial part of the component. It follows the agent`s behavior iteratively, and uses the acquired information about the impact of the executed actions on the game state, with the purpose of overwatching if there was a breach of validity of the game feature. The breach of validity is tested by the conditional statement.
3. Design and execution of the conditional statement: the conditional statement checks if the new game state is breaking the design (purpose) of the game feature (e.g. part of the map which should always be empty is now occupied).
4. Execution of the conditional statement`s condition: The conditional statement is the holder of the condition which is compared against the measured values. The condition must always return false, which states that the game feature is valid, because the condition testing for invalidity was not successful.

EXPERIMENT

- Game space of Tic-Tac-Toe was designed to operate in real-time. Real-time operation was achieved by not implementing any delays (hard-coded time slices) between the execution of two actions.

- Game feature (of choosing) for Tic-Tac-Toe: No-loss-strategy method (no errors/bugs present). Set of optimal strategies for an opponent never loosing a game (XCS algorithm plays against this method / opponent).



```
Presumption: method has full access to the game state / map
Function noLossStrategy(markerFriendly, markerOponnent):
1.  set counter = getCountGameStateNonEmptyCells()
2.  if counter = 0 then
3.     return new Position(5)
4.  end if
5.  set emptyCell = isTheCellEmpty(new Position(5))
    // opponent has a marker on the board, but not in the middle, so middle cell can
    // be taken
6.  if counter = 1 and emptyCell then
7.     return new Position(5)
8.  end if
    // if there is a possibility of a definite victory take advantage of it
9.  set takeAdvantageFriendly = takeAdvantageIfVictory(markerFriendly)
10. if not emptyPosition(takeAdvantageFriendly) then
11.    return takeAdvantageFriendly
12. end if
    // if enemy has definitive win in the next move, it has to be blocked
13. set takeAdvantageEnemy = takeAdvantageIfVictory (markerOponnent)
```

EXPERIMENT

```
if not emptyPosition(takeAdvantageEnemy) then
  return takeAdvantageEnemy
end if
if counter = 3 then
  if gameState[2] = markerOponnent and gameState[4] = markerOponnent then
    return new Position(1)
  end if
  if gameState[2] = markerOponnent and gameState[6] = markerOponnent then
    return new Position(3)
  end if
  if gameState[4] = markerOponnent and gameState[8] = markerOponnent then
    return new Position(7)
  end if
  if gameState[6] = markerOponnent and gameState[8] = markerOponnent then
    return new Position (9)
  end if
  // first sub-strategy against future scissors tactics
  if gameState[5] not markerOponnent
    and (      (gameState[1] = markerOponnent
               and gameState[9] = markerOponnent)
            or (gameState[3] = markerOponnent
               and gameState[7] = markerOponnent)
        ) then
    set list // holder of positions 2, 4, 6, 8
    return randomPositionFromList(list)
  end if
  // second sub-strategy against future scissors tactics
  if gameState[5] = markerOponnent then
    if gameState[1]  = markerOponnent
       or gameState[3]  = markerOponnent
       or gameState[7]  = markerOponnent
       or gameState[9]  = markerOponnent then
      set list // holder of positions 1, 3, 7, 9
      return randomPositionFromList(list)
    end if
  end if
end if
// random choosing of one of the corner cells: 1, 3, 7 or 9
if counter = 1 and not emptyCell then
  set list // holder of positions 1, 3, 7 or 9
  return randomPositionFromList(list)
end if
// with last four if statements mirroring corner positions are chosen
if gameState[1] = markerOponnent then
  if isTheCellEmpty(new Position(9)) then
    return new Position(9)
  end if
end if
if gameState[3] = markerOponnent then
  if isTheCellEmpty(new Position(7)) then
    return new Position(7)
  end if
end if
if gameState[7] = markerOponnent then
  if isTheCellEmpty(new Position(3)) then
    return new Position(3)
  end if
end if
if gameState[9] = markerOponnent then
  if isTheCellEmpty(new Position(0)) then
    return new Position (0)
  end if
end if
end function
```

EXPERIMENT

- The experiment consisted of two parts
  - The first part: the game space of Tic-Tac-Toe was comprised of only the valid game feature, acting as a test for XCS to confirm the validness of the no-loss-strategy. When playing against such an opponent, the XCS algorithm must not win.
  - The second part: intentional bugs were introduced into the game feature, making them invalid, which acted as a test of how successful the XCS was in finding those bugs.

- The purpose was to test the XCS algorithm to see if it discovered successfully, (through gameplay), that the method was not operating to its no-loss specifications when the bugs were present.

```
if resultOfTheGame not defeat and resultOfTheGame not tie then
        // game feature is not valid
end if
```

EXPERIMENT

ID of the bug	***Line numbers missing from the method (starting and ending positions)***	***Short description of a bug***
1.	6-8	Opportunity of putting a marker in position five is not taken, if an opponent already has one marker on the board and it is not occupying number five.
2.	9-12	Opportunity of a certain victory in the next move is not taken.
3.	13-16	Opportunity of preventing a certain victory for the opponent is not taken.
4.	18-29	Opportunity to prevent the opponent creating a scissor in the next move is not taken.
5.	30-33	Opportunity to prevent the opponent from any future scissor tactics (during the next moves) is not taken.
6.	34-38	Same description as before.
7.	40-43	Opportunity to block corners (lowering the search space and possibility of wins for the opponent) if there is only one marker on the map that is not taken.
8.	44-62	Similar description as before, but independent of the number of markers already present on the map.

EXPERIMENT

- Both experiments were executing blocks of games, with one block consisting of one million full-game playouts.

- The first part of the experiment was run by executing one block of games.

- The second part was run on two hundred blocks for each of the bugs intentionally included into the game feature.

EXPERIMENT

During the experiment, the following data were acquired (per each block):

- the consecutive number of the game in which the irregularity/invalidness of the game feature was first discovered,
- the number of all confirmations of the invalid game feature, and
- the time needed for completion of each block (the time measurements provided us with information if it was possible to validate the game space in real-time).

The upper bound of time, where the real-time measurement during one block execution is still acceptable, was set to a reasonable one minute.

RESULTS – FIRST PART

- Intact No-loss-strategy method (our game feature) on one million playouts (1 block) never resulted in a win for the XCS algorithm .

- Meaning, game feature was always valid (as intended).

- Note: we also made additional test of the method with one million playouts by a basic MCTS (UCT) reinforcement learning algorithm.

RESULTS – SECOND PART

ID of the bug	***Consecutive number of the game where XCS first confirmed invalid game feature [min. cons. num. of most succ. block, max. cons. num. of least succ. block] (The lower the values the better.)***	***The number of all confirmations of the invalid game feature in the block [min. num. conf. of least succ. block, max. num. conf. of most. succ. block] (The higher the values the better.)***	***Time needed to execute the block in seconds [min. time. of most. succ. block, max. time. of least succ. block] (The lower the values the better.)***
1.	[1, 1.784]	[38.413, 60.621]	[32,651, 33,893]
2.	[45, 109.330]	[135, 1200]	[29,737, 31,706]
3.	[1, 2]	[920.156, 930.109]	[23,963, 25,699]
4.	[1.259, 504.222]	[4, 339]	[30,819, 33,267]
5.	[20, 17.783]	[1.329, 3.240]	[30,371, 33,191]
6.	[1, 91.558]	[65, 154]	[30,154, 33,132]
7.	[24, 22.855]	[254, 470]	[29,945, 31,553]
8.	[11, 13.322]	[3.701, 7.616]	[31,256, 32,602]

CONCLUSION

- The consecutive numbers and the numbers of all confirmations were quite varied between the different inserted bugs.

- Some bugs are more common to be found, and some occur rarely during the game.

- Meaning, they can only be found when an XCS algorithm is traversing through the niche parts of the game (i.e. only in a few sub-search-spaces), and that can sometimes take many tries.

- The XCS algorithm identified all the bugs successfully, regardless of them occurring in common or niche search space paths.

FUTURE WORK

- We want to test our methodology of Game Feature validation with the XCS algorithm on game spaces which include maps of higher resolution (e.g. 32 x 32 cells), game actions with more diversity (i.e. durative game actions and simultaneous moves), and with more than one type of game unit available.

- Incorporating game actions with higher diversity in the XCS algorithm will be challenging.

- We envision creating a game agent, which has its internal structure designed for the operation of multiple XCS algorithms, and will be capable of handling all the game units on the map simultaneously, each with its own distinct actions.

- It will also be necessary for a game agent to incorporate more complex syntactic and semantic validations of classifiers, as well as a game state evaluation and reward system of higher complexity.

Thank you for your time. Please join the discussion.