Adaptive Difficulty

*How to modify the adaptive difficulty*

This document describes how to and what can be modified with respect to the adaptive difficulty and progreesion for both the tower defence and writing formal specifications part of the game. Most of the adaptive difficulty processes are present in the *adaptiveDifficultyScript.js* file present in the *GameServer* folder. This JavaScript file can contain many methods to modify the difficulty, which we will list below. The description of these method will use the following format:

|  |  |
| --- | --- |
| Method | Parameters |
| Description of the method. | |
| Return type | Default value |

The default value will be used incase the the method is missing or an error occurs during the process. Because of the way values are interpreted by the JavaScript engine, return values that are specified as an int can be float

We will first list the methods related to tower defence, then those related to writing formal specifications and progression, then the processing method, finally we will list the data present in the objects send as parameters.

A barebones implementation containing most of the features below is given in *basicADS.js*.

## Tower Defence

|  |  |
| --- | --- |
| getWaveSparkAmount | data |
| Returns the amount of sparks that should spawn in the next wave. | |
| int | 10 |

|  |  |
| --- | --- |
| getWaveSparkHealth | data |
| Returns the amount of health the sparks should have in the next wave. | |
| int | 30 |

|  |  |
| --- | --- |
| getWaveSparkSpeed | data |
| Returns the base speed the sparks should have in the next wave.  This base speed is slightly modified in the game to make the sparks move at slightly different speeds. | |
| int | 64 |

|  |  |
| --- | --- |
| defaultWaveSparkSpawnTime | data |
| Returns the amount of milliseconds between the spawning of sparks, limited to one per frame. | |
| int | 1000 |

|  |  |
| --- | --- |
| getPrePercentageMultipliers | data |
| Returns modifiers for the weights of the types of sparks to spawn for the pre condition path. | |
| [float, float, float, float] | [1.0f, 1.0f, 1.0f, 1.0f] |

|  |  |
| --- | --- |
| getPostPercentageMultipliers | data |
| Returns modifiers for the weights of the types of sparks to spawn for the post condition path. | |
| [float, float, float, float] | [1.0f, 1.0f, 1.0f, 1.0f] |

|  |  |
| --- | --- |
| getSpecialSparkSpawnPercentage | data |
| Returns percentages on what special sparks to spark. | |
| [float, float] | [0.01f, 0.01f] |

## Formal Specifications and Progression

|  |  |
| --- | --- |
| getMinimumDifficulty | data |
| Returns the minimal difficulty for the next problem. | |
| int | 1 |

|  |  |
| --- | --- |
| getMaximumDifficulty | data |
| Returns the maximum difficulty for the next problem. | |
| int | 999999 |

|  |  |
| --- | --- |
| isFinalProblem | data |
| Returns the next problem is the final problem. | |
| bool | True (The prevent endless gameplay, due to crashes or misprogramming of the script) |

**In the following methods with *optionalityString* we mean a string which is either “have”, “notHave” or “optional” (case insensitive).**

**“have” implies that the next problem must have that property,**

**“notHave” implies that the next problem must not have that property and**

**“optional” implies it does not matter whether the next problem has that property.**

|  |  |
| --- | --- |
| getHasForAll | data |
| Returns whether the next problem should have a for all quantifier in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasExists | data |
| Returns whether the next problem should have a exists quantifier in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasArrays | data |
| Returns whether the next problem should have arrays in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasEquality | data |
| Returns whether the next problem should have equality operators (==, !=) in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasLogicOperator | data |
| Returns whether the next problem should have logic operators (&&, ||, !) in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasRelationalOperator | data |
| Returns whether the next problem should have relations operators (>, <, >=, <=) in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasArithmetic | data |
| Returns whether the next problem should have arithmetic operators (+, -, \*, /, %) in the teacher solution. | |
| optionalityString | “optional” |

|  |  |
| --- | --- |
| getHasImplication | data |
| Returns whether the next problem should have implication in the teacher solution. | |
| optionalityString | “optional” |

## Processing

|  |  |
| --- | --- |
| processPreResponse | data, response |
| Gets called when a pre condition is submitted by the player to process the response from the equivalence checker. | |
| void | None |

|  |  |
| --- | --- |
| processPostResponse | data, response |
| Gets called when a post condition is submitted by the player to process the response from the equivalence checker. | |
| void | None |

|  |  |
| --- | --- |
| processWaveData | data, waveData |
| Gets called when a wave is completed to process the data about the wave. | |
| void | None |

|  |  |
| --- | --- |
| processProblemStatistics | data, problemStatistics |
| Gets called whenever a problem is started to process the statistics about that problem. | |
| void | None |

|  |  |
| --- | --- |
| newProblem | data |
| Gets called whenever a new problem is started. | |
| void | None |

## Objects

The following methods are accessible from their respective object should not be programmed in the JavaScript file.

### Data

|  |  |
| --- | --- |
| getPreMistakeCount | |
| Returns the amount of mistakes the player made for the pre conditions of the current problem. A mistake counts as having made a change to the precondition submitted and the send precondition being incorrect. | |
| int | |

|  |  |
| --- | --- |
| getPostMistakeCount | |
| Returns the amount of mistakes the player made for the postconditions of the current problem. A mistake counts as having made a change to the postcondition submitted and the send postcondition being incorrect. | |
| int | |

|  |  |
| --- | --- |
| getTeacherProblemDifficulty | |
| Returns the difficulty of the teacher problem. | |
| int | |

|  |  |
| --- | --- |
| getFeatureUsage | |
| Returns an array of containing the amount of times certain features have been used in submissions. | |
| int[] | |

|  |  |
| --- | --- |
| getFeatureMask | |
| Returns an array of whether certain features are used in the current problem. | |
| boolean[] | |

### 

|  |  |
| --- | --- |
| getQuestionAmount | |
| Returns the least amount of problems the teacher wants the player to solve before finishing the game. | |
| int | |

|  |  |
| --- | --- |
| getTeacherProblemHasForAll | |
| Returns whether the teacher solution contains the for all quantifier. | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasExists | |
| Returns whether the teacher solution contains the exists quantifier. | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasArrays | |
| Returns whether the teacher solution contains arrays. | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasEquality | |
| Returns whether the teacher solution contains equality operators (==, !=). | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasLogicOperator | |
| Returns whether the teacher solution contains logic operators (&&, ||, !). | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasRelationalComparer | |
| Returns whether the teacher solution contains relational comparers (>, <, <=, >=). | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasArithmetic | |
| Returns whether the teacher solution contains arithmetic operators (+, - , \*, /, %). | |
| bool | |

|  |  |
| --- | --- |
| getTeacherProblemHasImplication | |
| Returns whether the teacher solution contains implications. | |
| bool | |

### Response

|  |  |
| --- | --- |
| isEquivalent | |
| Returns whether the conditions are equivalent. | |
| bool | |

|  |  |
| --- | --- |
| getPreFeedback | |
| Returns feedback about the precondition in the following form:  [0]: (T,T) Whether there exists input such that both the players precondition and the teachers precondition is true.  [1]: (T,F) Whether there exists input such that the players precondition is true and the teachers precondition is false.  [2]: (F,T) Whether there exists input such that the players precondition is false and the teachers precondition is true.  [3]: (F,F) Whether there exists input such that both the players precondition and the teachers precondition is false. | |
| [bool, bool, bool, bool] | |

|  |  |
| --- | --- |
| getPostFeedback | |
| Returns feedback about the postcondition in the following form:  [0]: (T,T) Whether there exists input such that both the players postcondition and the teachers postcondition is true.  [1]: (T,F) Whether there exists input such that the players postcondition is true and the teachers postcondition is false.  [2]: (F,T) Whether there exists input such that the players postcondition is false and the teachers postcondition is true.  [3]: (F,F) Whether there exists input such that both the players postcondition and the teachers postcondition is false. | |
| [bool, bool, bool, bool] | |

### ProblemStatistics

|  |  |
| --- | --- |
| getAverageWavesNeeded | |
| Returns the average amount of waves players needed to complete the problem. | |
| float | |

|  |  |
| --- | --- |
| getAveragePreMistakeCount | |
| Returns the average amount of mistakes players made on the precondition. A mistake counts as having made a change to the precondition submitted and the send precondition being incorrect. | |
| float | |

|  |  |
| --- | --- |
| getAveragePostMistakeCount | |
| Returns the average amount of mistakes players made on the postcondition. A mistake counts as having made a change to the postcondition submitted and the send postcondition being incorrect. | |
| float | |

### WaveData

|  |  |
| --- | --- |
| getScore | |
| Returns the total amount of score the player has at the end of the wave. | |
| int | |

|  |  |
| --- | --- |
| getDeltaScore | |
| Returns the amount of score the player gained or lost during the wave. | |
| int | |

|  |  |
| --- | --- |
| getMoney | |
| Returns the amount of money the player has at the end of the wave | |
| int | |

|  |  |
| --- | --- |
| getHealth | |
| Returns the amount of health the player has at the end of the wave. | |
| int | |

|  |  |
| --- | --- |
| getTowerCount | |
| Returns the total amount of towers the player currently has on the field. | |
| int | |

|  |  |
| --- | --- |
| getPreSpawned | |
| Returns the amount of different types of sparks that spawned for the precondition as follows:  [0]: (T,T) Green sparks that are not marked.  [1]: (T,F) Green sparks that are marked.  [2]: (F,T) Red sparks that are not marked.  [3]: (F,F) Red sparks that are marked. | |
| [int, int, int, int] | |

|  |  |
| --- | --- |
| getPrePassed | |
| Returns the amount of different types of sparks that passed the precondition path as follows:  [0]: (T,T) Green sparks that are not marked.  [1]: (T,F) Green sparks that are marked.  [2]: (F,T) Red sparks that are not marked.  [3]: (F,F) Red sparks that are marked. | |
| [int, int, int, int] | |

|  |  |
| --- | --- |
| getPostSpawned | |
| Returns the amount of different types of sparks that spawned for the postcondition as follows:  [0]: (T,T) Green sparks that are not marked.  [1]: (T,F) Green sparks that are marked.  [2]: (F,T) Red sparks that are not marked.  [3]: (F,F) Red sparks that are marked. | |
| [int, int, int, int] | |

|  |  |
| --- | --- |
| getPostPassed | |
| Returns the amount of different types of sparks that passed the postcondition path as follows:  [0]: (T,T) Green sparks that are not marked.  [1]: (T,F) Green sparks that are marked.  [2]: (F,T) Red sparks that are not marked.  [3]: (F,F) Red sparks that are marked. | |
| [int, int, int, int] | |

|  |  |
| --- | --- |
| getMoneySpent | |
| Returns the amount of money the player spent during the wave as follows:  [0]: Spent on towers.  [1]: Spent on blocks for formal specifications | |
| [int, int] | |

|  |  |
| --- | --- |
| getTimeSpent | |
| Returns the amount of time the player spent during the wave as follows:  [0]: Before the wave spawned.  [1]: In block building mode. | |
| [int, int] | |