BWBrothers: System manual

2018 Tomáš Varga

Content

List of Tables 3

1 Python server: BWBigBrother 4

1.1 Dependencies 4

1.2 App.py 4

1.3 SCUtilsOnline.py 6

2 Java module: BWSmallBrother 8

2.1 Dependencies 8

2.2 Main.java 8

2.3 BWSmalllBrother.java 8

2.4 ServerUCType.java 10

2.5 CustomClasses.java 12

List of Tables

Tab. 1: Global variables app.py 5
Tab. 2: Web methods app.py 5
Tab. 3: General methods 6
Tab. 6: General methods ScUtilsOnline.py 7
Tab. 7: Properties of BWSmallBrother 9
Tab. 8: Methods of BWSmalBrother 10
Tab. 9: Properties of ServerUCType 10
Tab. 10: Methods of ServerUCType 11
Tab. 11: Custom classes 12

1 Python server: BWBigBrother

BWBigBrother is a Python server used primarily as a prediction machine for game StarCraft. It's using REST API and JSON for communication with users.

1.1 Dependencies

- Python: 2.7+
- Libraries:
  - Flask 0.10.1,
  - Numpy 1.10.4,
  - pandas 0.17.1,
  - scikit-learn 0.19,
  - matplotlib.pyplot 1.3.1.

1.2 App.py

It's a main script for the Python server and it contains all the logic behind training models, predicting models and communicating with the game through the BWSmallBrother module. Data coming here is sent to the other scripts where it is processed and returned here. After that, the models learn to make predictions of the final outcome of strategies based on this processed data.

GLOBAL VARIABLES	DESCRIPTION	TYPE
IP	Server IP address.	String
EACH_FRAME	Send to server each frame - default 240, can be overwrite from settings.ini.	Integer
TRAINING_DATA	Path to the training dataset.	String
MODEL_DIRECTORY	Directory where the software should look for models.	List
MODEL_FILES_NAME	Models' name and path, from where the models should be imported and exported.	List
MODEL_COLUMNS_FILES_NAME	Models' path from where the models columns should be imported and exported.	List
MODEL_COLUMNS	Column names.	List

CLF	Multiple trained classifiers.	list
HEADERS	Attributes for each model.	dict(List)

Tab. 1: Global variables app.py

Web methods are functions dedicated to communicate with the "outside" world via REST API. Here we can send the data with JSON or HttpRequest, upon different function will be done. They return results either in HttpBody format or in JSON.

WEB METHODS	REST API ENDPOINTS	DESCRIPTION	INPUT	OUTPUT	HTTP METHOD
PREDICT	/predict	JSON from the game is sent here and processed afterwards. The outcome of predictions is sent back to the game.	JSON	JSON	POST
TEST	/test	Function for testing the model. It loads JSON file – test.json, sends it to predictModels function and returns correct results. If the models were trained correctly, it will return JSON with the outcome.	REQUEST	BODY	GET
TRAIN	/train	Training all the models upon receiving the request from the user.	REQUEST	JSON	GET
WIPE	/wipe	After sending the request on this function, each .pkl file is removed and each model is reset.	REQUEST	BODY	GET

Tab. 2: Web methods app.py

Methods listed in Tab. 3 serve as additional methods to help either with models training or predicting with them.

general methods	DESCRIPTION	PARAMETERS
predictModels	Predicts the outcome of actions for each model.	Replay, PlayerBase
trainModel	Trains models. after user sends request on /train endpoint.	index, clf, model_columns, model_columns_file_name, headers, modelName
split_dataset	It splits dataset to train and test samples in specified ratio and with feature/target headers.	dataset, train_percentage, feature_headers, target_header
gradient_boosting	Gradient boosting classifier with best parameters for each strategy.	index, features, target

Tab. 3: General methods

1.3 SCUtilsOnline.py

The script is used to process the data coming from the game, so that the models can use them to make predictions of the strategies. It includes the same global variables as the script gsp.py. Both of them need to have loaded .csv files available in order to work properly.

METHODS	DESCRIPTION	PARAMETERS
COUNTNUMBEROF	Based on the parameters it counts units in columnName and checkInList starting from defaultNumber.	data, columnName, checkInList, defaultNumber, countWorkers
IDENTIFYRACE	Identifies the race based only on worker IDs.	data
GETCOORDINATES	Gets coordinates to be used for later calculation.	data
FINDSTARTLOCATION	Finds possible start locations.	data
FINDSCOUTUNIT	Finds scout unit in the beginning of the game.	data
SCOUTINTHEGROUP	Identifies scout unit in the row.	data, columnName, checkInList, replayID
ISNEARNEMY	Tries to determine if the player's unit is outside of his own base.	data, scoutGroup, base1, base2, replayID, timeStop
EXPFEATURE	Finds feature e.g. unit, specified by columnName and checklist then determine if append 0 or 1.	data, columnName, checkInList, replayID, cooldownActiv
RATIO	Calculates ratio specified by parameters.	data, columnName, checkInList
GETUNITGROUPIDS	Searches IDs of the units in checkInList and TargetID. Then it collects and returns the group IDs of these units.	data, checkInList
PREPROCESS	Main function inside the script. It applies each function that was mentioned before on the original data coming from the game. At the end it creates data suitable for models.	nData

Tab. 4: General methods ScUtilsOnline.py

2 Java module: BWSmallBrother

Java module to the game that communicates with BWAPI and server. All the game data are aggregated here into actions and sent to the server.

2.1 Dependencies

- Java version: 1.8.0_144,
- BWAPI version: 4.1.2,
- Libraries:
  - bwmirror 2_5.jar – BWAPI Java interface,
  - ini4j.jar – easy work with the .ini files,
  - httpcore.jar and httpclient.jar – Apache library simplifies works with HTTP,
  - gson.jar – Google library making it easier to wrap up JSON,
  - commons-logging.jar – Apache dependencies.

2.2 Main.java

The main purpose of this script is to create new BWSmallBrother object with function *run()*. It also serves as a main part of the Java module.

2.3 BWSmalllBrother.java

Script, where all the logic in the module is incorporated. In this script, the system also listens for BWAPI commands with inheriting DefaultBWListener class and each of the methods in this class.

PROPERTIES	DESCRIPTION	TYPE
MIRROR	BWAPI Mirror object - listens to the game states.	Mirror
GAME	Every information from the game.	Game
PLAYER	Player to be recorded.	Player
SEND_EACH_FRAME	When to send the JSON file to the server.	Integer
FRAMETORECORD	Counts send_each_frame.	Integer

STRATEGIES	The name of the strategy and text with status active or - .	Map<String, String>
ACTIONS	All actions from the recorded players.	List<Action>
ACTIONCOUNT	ActionID.	Integer
UNITGROUPID	UnitGroupID.	Integer
RENDSTARETGY	Where on screen should the game render text with predicted strategies's names.	integer
SNAPFRAMES	Each frame with unit order and position.	Map<Integer, FrameSnap>
UNITGROUPIDS	UnitGroupIDs.	Map<Integer, Integer>
INI	Settings. ini file.	Ini
PNAME	Player's name from .ini file.	String
BASE	Player's start location,	PlayerBase

Tab. 5: Properties of BWSmallBrother

The attribute ORIGIN in the Tab. 8 describes whether the function was made specifically for this software or it was already built inside the BWAPI. Note that each method in the table was modified, as BWAPI serves only as a link between the game and the software.

METHODS	DESCRIPTION	PARAMETERS	ORIGIN
RUN	It loads the .ini file, sets the default strategies text and starts to listen for the game data.	-	BWAPI
ONEND	When the match ends, the BWAPI informs the software about it. Hard reset of each property.	-	BWAPI
ONSTART	Loads player name from the .ini file, sets the default player, analyzes terrain.	-	BWAPI
ONFRAME	Main logic for the Java module. It starts to record each action of the player	-	BWAPI

	specified in the property Player.		
ADDNEWACTION	Adds new actions upon getting the ID from dictionary lookup, and from the parameters.	playerID,unitCommandType, orderType,unitGroupID,targetID, targetX,targetY, unitType, frame, race	CUSTOM
FINDUNITBACKINTIME	Finds the unit ID from previous frame.	Frame, myUnit	CUSTOM
PREDICTIONS	Sends all the actions and base as one object called predictobj. After that, the gson function converts it to JSON and HttpRequest sends it to the server. The server then responds and if a strategy was detected, the text in the game changes.	-	CUSTOM

Tab. 6: Methods of BWSmalBrother

2.4 ServerUCType.java

This class has four properties specified in Tab. 9. After creating an object of this class, each property creates a map – dictionary lookup with ID from JINBWAPI and BWMirror class type.

PROPERTIES	TYPE
SERVERUCTYPEIDS	Map<UnitCommandType, Integer>
SERVEROTYPEIDS	Map<Order, Integer>
SERVERUNTYPEIDS	Map<UnitType, Integer>
SERVERRACEIDS	Map<Race, Integer>

Tab. 7: Properties of ServerUCType

For example, when the system calls serverUCTypeIds.get(unitcommandtype) the program tries to lookup into this object and returns ID for unitcommandtype.

METHOD	DESCRIPTION	PARAMETERS
CREATEMAP()	Creates dictionary for UnitCommandType.	actionID, PlayerReplayID, UnitCommandTypeID, OrderTypeID, Unit
CREATEMAP2()	Creates dictionary for Order.	snapUnit, order, orderTargetPosition
CREATEMAP3()	Creates dictionary for UnitType.	cannon_rush, three_gateways, two_gateways, zergling_rush
CREATEMAP4()	Creates dictionary for Race.	candidates, freq

Tab. 8: Methods of ServerUCType

2.5 CustomClasses.java

Custom classes created to improve either the work with the software or the communication with the server. Classes like action, result, seq, predictobj are used to be converted either to or from JSON.

CLASS	DESCRIPTION	PROPERTIES
ACTION	The class is used later in the predictActions and sequentialMining functions.	actionID, PlayerReplayID, UnitCommandTypeID, OrderTypeID, Unit
FRAMESNAP	Data object, into which the system encodes every order and position of each unit.	snapUnit, order, orderTargetPosition
RESULT	Results coming from the server are transformed from JSON into this object.	cannon_rush, three_gateways, two_gateways, zergling_rush
PLAYERBASE	The location of the user's starting base is encoded into this object which is then transformed to JSON.	x,y
PREDICTOBJ	Both properties actions and base are used for transforming this object into JSON.	actions, base

Tab. 9: Custom classes