Testing of games through software agents managed by artificial neural networks

Miroslav Trankov¹, Emil Hadzhikolev², Stanka Hadzhikoleva³
¹,²,³ University of Plovdiv “Paisii Hilendarski”, Department of Computer Science, 236, Bulgaria Blvd. Plovdiv, Bulgaria
¹trankov86@gmail.com, ²emil.hadjikolev@gmail.com, ³stankah@gmail.com

Abstract. With the development of artificial neural networks, their use in different fields is increasing to optimize different activities. The availability of rich programming languages, various existing machine learning software libraries, along with the virtualization technologies has made intelligent technologies accessible not only to large corporations but also to individual programmers.

Game development is a complex process involving many specialists, experts in various fields. Single-game testing is an iterative process that covers different game development opportunities, different player abilities, as well as player interaction when working in a team. The question is, is it possible to use intelligent software agents in the game testing process in order to optimize the human, time and financial resources needed?

This article introduces our attempt to create an intelligent autonomous agent that plays the Brawl Stars game, managed by an artificial neural network. Initially, the agent is trained by collecting frames from a game that is being played by an experienced player. The frames are then processed and analyzed, and a neural network is trained, which directs the agent to conduct an autonomous game. The developed agent was used in two experiments that confirmed the great potential for the use of artificial neural networks in game testing.

1. Introduction

The fast-growing gaming industry imposes new requirements and rules on game development. The process involves different professionals - designers, programmers, artists, testers, experts in different fields, etc. Artificial intelligence is also used to improve the consumer experience, in its various forms - through autonomous agents [1], neural networks [2], game software templates [3] and so on.

The implementation of AI in games is not a novelty. The first successful attempts to deploy an autonomous agent in a game date back to the early 1970s, and were conducted with the well-known arcade games Computer Space-1971 and Space Invaders-1978. The developers have realized that for better and unique user experience, games must interact with the player independently and adequately, changing the game environment, objects, activities and processes within it. The emergence of AI as an independent field in computer science has an impact on all other industries. In the 1980s, companies actively invested in building expert systems for solving specific problems [4]. A big success in the 1990s was the IBM-created AI Deep Blue chess game application [5]. The second improved version of Deep Blue in 1997 won the match against World Chess Champion Gary Kasparov. Another big challenge for game developers is to develop an application for another strategy game with simple rules - Go. In October 2015, the DeepMind-developed AlphaGo program [6] defeated Fan Hui, the
European Go Champion, five times in a five-game match. In March 2016, an upgraded version of AlphaGo wins four-on-one victories against Go's leading master of the world, Lee Sedol.

Behind the rise of AI applications in recent decades have been major technology and software companies such as Facebook, Google, Amazon, OpenAI, IBM and Intel [7]. They create multiple software libraries for AI and Machine Learning (ML). Most libraries are open source and some have versions in different programming languages, such as Python, C / C++, C #, Java, and more. Many experts consider Python as the most preferred language for AI and ML due to of its built-in big data capabilities, visualization tools, syntax support for various programming styles (procedural, object-oriented, functional), platform independence etc. That's why Python also has the most AI and ML libraries like TensorFlow [8], Keras [9], Sci-kit Learn [10], Theano [11], PyTorch [12] and so on.

Open source AI and ML libraries provide the opportunity to leverage more powerful and better ways to build their own AI applications, not only to large companies but also to individual developers. Building an artificial neural network for interacting with digital game is nowadays quite accessible and interesting task for developers dealing with AI - GTA 5 [13], FIFA 18 [14], CS: GO [15] and others.

Testing games in the process of their implementing is an important task that can be automated to a great extent by using artificial intelligence techniques. This article describes our experience in creating an automated game playing software agent using neural networks for training and self-study. The object of the experiment is, the well-known among the youngers, multiplayer, strategy and shooting game called Brawl Stars [16]. The agent and the game are running in the Operating System (OS) emulator. The agent's actions are guided by an artificial neural network that monitors the game's development through reading game frames from the screen, and its reactions are transmitted to the game via the OS emulator.

2. Game interaction with autonomous software agents

There are two main approaches for an autonomous software agent to communicate with a computer game: direct and indirect interaction.

In direct interaction (figure 1), the software agent usually communicates with the game through the program interfaces it supports. For security and copyright reasons, not all applications support this feature. Another option for direct interaction is software integration into the code by manipulating the game code. The difficulties of implementing such an agent are related to learning foreign code without standardized programming interfaces and/or programming techniques.

Indirect interaction (figure 2) is an approach in which the software agent communicates with the environment in which the games are played, receives information from the environment and through it influences the game. Agents can be located both inside and outside the environment. Programming interfaces and standardized tools come from the environment and are not game-specific. Many existing emulators for different operating systems could be used as an environment. A major drawback of indirect interaction is the inability to use specialized game control capabilities. The advantage is the ability to build universal technology for automated management of games and various other applications, whether or not they provide standard mechanisms.
The inability to directly control the game through indirect interaction requires the search for new approaches. In 2017, Harrison Kinsley [13] used a convolutional neural network to navigate a player in GTA 5. The convolutional neural network uses real-time footage of the game and, after training, begins to play independently. The approach offered by Kinsley is subsequently implemented and adapted across a wide range of games. The main steps in its implementation are (figure 3):

1. Accumulate enough frames and associated keystrokes from a real player.
2. Training and testing of artificial neural network with the collected frames.
3. Real-time neural network operation using current frames.

3. Model of an autonomous Brawl Stars game agent

Using the approach suggested by Kinsley, we implement an autonomous software agent for the Brawl Stars game [16] (figure 4). The popular among the youngsters multiplayer strategy and shooting game Brawl Stars is an application for iOS and Android-based devices. In the experiments, the game was started in an Android emulator on Windows. The agent is also running on Windows and communicates indirectly with the game using the Python language capabilities to manage Windows input/output operations.

Important factors for the implementation are the player control capabilities and the ability of the agent to be trained to perform automatically. It is controlled by several on-screen joysticks:

- **blue** (left) - to navigate the playing field; it is enough to keep it in the right direction; if the opponent's player is within range of the player's weapon, **the firing is automatically activated**;
- **red** (right) - pressed to start **precise aiming**, and when released, the charge is fired;
- **yellow** - appears when the **Super charge** accumulates and disappears after it is activated when pressed.

A limitation that exists in firing the weapon is that after three consecutive shots, it is necessary to wait 3 seconds to produce new shots.

Considering the control mechanisms, **the software agent management model in the first variant developed by us comes down to training and using ANN to navigate the character. In this case, the activity of the neural network is a simulation of pressing the blue screen joystick to move the player in different directions - without movement, left, right, back, forward, as well as intermediate left-up, left-down, right-up, right-down - determined by the location of both the opposing players and teammates.** The restriction that we have imposed on firing is that the shots should be at regular
intervals, but not less than 2 seconds. For better results, it is appropriate for the software agent to implement and integrate two neural networks that operate simultaneously but independently of each other: one for navigation and the other for aiming and firing.

**The main modules of the software agent** that can work independently in different periods (figure 5) are:

- **The game frames engine**, with its Frames collector and Frames analyzer sub-modules, are responsible for the collection and initial processing of frames during a person-to-person game;

- **AI training engine**, through the Serialization and Models training sub-modules, processes the collected frames and through NN creates models that can be used in automated control.

- **Player engine** - uses the generated models to play in real-time.

The first major activity of the software agent is to collect enough screenshots during the course of a person's game. The Frames collector module is responsible for recording screenshots of the game. Its **main input parameters are screen size and location, determined by the top-left pixel**. Since the game has many different terrains and characters to play with, the terrain and the character are also defined as input parameters. In our experiments, the game Brawl Stars was launched in OS Emulator, covering a window of 800x600px, the upper left corner of which is (0, 0). The number of screenshots required is determined by the accuracy we want to achieve in neural network training. E.g. our experiments have shown that in the presence of 300 screenshots for each of the possible movements, an accuracy of 99.63% is achieved, and for 200 screenshots - about 95%. As it became clear, the training of the NN is done on the basis of the frames collected during the actual game. For better results, a gamer who plays the game during the training period of the NN must be experienced and skillful.

The Frame analyzer analyzes and normalizes the collected screenshots, and divides them into nine different folders - for each of the nine movements allowed in the game. Building an effective neural network model requires careful consideration of the network architecture as well as the format of the input data. The most common image data entry parameters are the number of images, the height and width of the image, the number of channels, and the number of pixel levels. Usually, we have 3 data channels corresponding to colors - Red, Green, Blue (RGB). Pixel levels are usually in the range [0,255]. In order to improve the accuracy of the model, the number of photos must be the same in each of the directories. The same number of images for each category is a process of balancing and evenly distributing the input data, which guarantees that the model will not be misled in its training [17]. To test the accuracy of the training model created, a number of random images are extracted.

---

**Figure 5. Software Agent Architecture and Core Modules**
In the Serialization module, screenshots, along with their metadata, are saved in stream (HDF5) format for faster access. HDF5 is a cross-platform technology designed to record and quickly access complex big data [18].

For model training, we use a convolutional neural network (CNN) in the Models training module. CNN [19] is a deep learning algorithm that accepts images as input and then uses different features/aspects in the image to distinguish different objects. The advantage of CNN is the capabilities they offer for image processing and object classification [20]. By examining the color characteristics of two-dimensional images, in particular the frames of the game, CNNs transform them into three-dimensional objects. In addition, CNN can automatically recognize predefined categories such as different types of animals, faces, objects etc. Suitable for research in the development of such applications is the General Game Playing algorithm Monte Carlo Q-learning [21].

The typical CNN has two parts:

- **A convolutional base that is made up of layers.** The main purpose of the convolutional base is to generate characteristics [22]. Each image can be viewed as a set of specific features (structures): points, ovals, lines, edges that form objects. Features can be related to the movement of objects in a sequence of images, of different sizes and shapes. The concept of features is very general and defining them in a particular computer image classification task can be highly dependent on the particular problem.

- **A classifier** that is usually composed of fully coupled neural layers. The main purpose of the classifier is to classify the image based on the detected characteristics (filters). Any classifier of machine learning algorithms such as Logistic Regression, SVM, Random Forest, Decision Tree, etc. can serve as a classifier with a different performance for different tasks.

The chosen approach for training the CNN in this particular implementation is transfer learning using only gathered game frames [23]. Here, a pre-trained network on a particular problem is used to solve another similar problem. Implementing a ready-made model is a smart solution that saves the need to collect a large array of training data, create your own implementation, and have the serious processing power for NM training. Therefore this gives the advantage to use only frames from the game across the whole process of evaluation (input, training and testing). In this case, in working with images, we can use neural network architectures such as VGG16, VGG19, Xception, ResNet50, InceptionResNetv2 and MobileNet, Inception-v3 [24] in order to speed up the learning process. There are some trained on ImageNet database [25] and available in the Keras [9] library in Python. In our experiments, we use the Inception-v3 model [26]. When configuring it, we set the frames collector images that form the user layer, as well as a classification algorithm for determining the nine movements of the player. **The characteristics of CNN's convolutional base are extracted from the user layer of the model.** We have chosen Logistic Regression as the most suitable classification algorithm.

Here comes a question regarding how sufficient could be a pre-trained model created only with self-gathered data? There are plenty of academic researches proving how easy and fast are machine learning approaches using transfer learning [27], [28].

In the experiments performed using Logistic Regression, at 459 frames the classification accuracy was 76.67%, while at 2700 frames the model accuracy increased to 99.63%. The time it took to train a pre-trained model using 459 frames was 17 minutes and for 2,700 frames was approximately 53 minutes. This means that time spent on training is approximately 1 sec. for a frame. The hardware specification of the machine used is listed below.

The Player module is an autonomous player that plays based on the models stored in the Storage. The player navigates through the trained neural network. According to the chosen approach, shooting at opponents takes place at regular intervals by simulating mouse clicks at a point on the screen with certain coordinates.

The main scheme on which the base modules work (figure 6) is the following:

- The user Gamer starts the game and plays for a certain period of time in order to train the autonomous agent.
Throughout the game, the **Software agent** collects screenshots through the **Game frames engine**, which analyzes, sorts, and saves the data in **Storage**, along with automated metadata.

The NN training process can be started by the **AI training engine** when enough screenshots are collected, but it can also be started by the user at any time. Models created during training are used by the **Player engine**, which can be started automatically or by a user.

![Diagram of software agent and sequence of implementation]

**Figure 6.** Main activities of the software agent and sequence of their implementation

### 4. Implementation and experiments

As part of the study, an autonomous Python language agent was developed. The application files are available at [https://github.com/snooty7/BrawlStars](https://github.com/snooty7/BrawlStars). The following specialized libraries have been used for the development:

- grab-screen - for taking screenshots;
- numpy - the main library for scientific computing;
- h5py - for working with HDF5 binary data format;
- scikit-learn - a powerful library of different machine learning methods;
- keras - neural-network library;
- pywinauto - for automate the Microsoft Windows GUI, etc.

As Android emulator that plays the game Brawl Stars, we use Nox [29], but other free or paid emulators like Bluestacks, MeMu, and more can be used. The basic parameters of the hardware system on which the experiments were conducted are the following: Intel Core i5-4200M processor (2-core, 2.50 - 3.10 GHz, 3 MB cache), 8 GB RAM (2x4096 MB) - DDR3, 1600 Mhz, video AMD Radeon HD 8750 M Card (2GB) and Kingston 480GB SSD Hard Drive. At peak times, the CPU load reached...
75% and running temperature of 62° C. No breaks or other problems with the application modules were observed.

The experiments in the Brawl Stars game were performed with a character called Colt, at an Excellent Escapade terrain in Gem Grab mode, where the team collects gems. The team wins by collecting 10 gems and holding them for 15 seconds. In the current implementation, the Agent player does not participate in the diamond collection but assists the team in seeking to eliminate the opposing players. The strategy that Gamer plays in order to train the neural network more effectively is to dynamically navigate the terrain and make it difficult for players on the opposing team to attempt to collect diamonds.

The results of the two experiments are as follows:

- The neural network is trained by a gamer with a personal level of 3,300 cups. The Agent player has won 6 of 10 games.
- The neural network is trained by a gamer with 320 cups. Agent player won 8 of 10 games.

By themselves, these results do not indicate the level of play of the trained agent, since the end result depends on the skills of teammates and opponents, as well as the game strategy through which the training was conducted. The result we found in the game is that Agent player manages to make relatively correct decisions.

5. Conclusion
The use of different AI technologies in games in order to create more exciting and realistic consumer experiences establishes new, ever-increasing demands on game development. Testing a game in the process of its development requires a lot of time and human resources. Using AI can optimize this activity.

This article presents a study, based on using an autonomous software agent to play as a team in the Brawl Stars game. The agent is trained by collecting, researching and analyzing frames from a real human-player game. He then plays alone, managed by a neural network.

The experiment shows that the use of an autonomous software agent can simulate a player and facilitate the software testing of a multiplayer network game. The implementation performed is suitable for testing any type of navigation games - car racing, arcade games with airplanes, bikes and other vehicles.

Acknowledgements
The work is partly funded by the SP19-FMI-004 project at the Research Fund of the University of Plovdiv "Paisii Hilendarski".

References
[1] Nareyek A 2001 Review: Intelligent Agents for Computer Games Lecture Notes in Computer Science, vol. 2063. Springer, Berlin, Heidelberg.
[2] Charles D and McGlinchey S 2004 The past present and future of artificial neural networks in digital games.
[3] Treanor M, Zook A, Eladhari M et al. 2015 AI-based game design patterns Proc. Foundation of Digital Games 2015 Conf.
[4] Hart P 1982 Directions for AI in the eighties ACM SIGART Bulletin Issue 79 January 1982.
[5] Deep Blue online: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue.
[6] AlphaGo DeepMind online: https://www.deepmind.com/research/case-studies/alphago-the-story-so-far.
[7] Patrizio A 2019 Top 45 Artificial Intelligence Companies Datamation.
[8] TensorFlow online: https://www.tensorflow.org.
[9] Keras: The Python Deep Learning library online: https://keras.io.
[10] Scikit-learn, Machine Learning in Python online: https://scikit-learn.org/stable.
[11] Theano online: https://github.com/Theano/Theano.
[12] PyTorch online: https://pytorch.org.
[13] Reading game frames in Python with OpenCV - Python Plays GTA V online: https://pythonprogramming.net/game-frames-open-cv-python-plays-gta-v/.
[14] Trivedi Ch 2018 Building a Deep Neural Network to play FIFA 18 Towards data science online: https://towardsdatascience.com/building-a-deep-neural-network-to-play-fifa-18-dec54d45e675.
[15] Balsys R 2019 Testing YOLO v3 CS:GO Keras object detection online: https://pylessons.com/YOLOv3-CSGO-detection/.
[16] Brawl Stars online: https://brawlstars.com.
[17] Krizhevsky A Sutskever I and Hinton G 2012 ImageNet Classification with Deep Convolutional Neural Networks Advances in Neural Information Processing Systems 25 (NIPS 2012).
[18] The HDF5 Library & File Format The HDF Group online: https://www.hdfgroup.org/solutions/hdf5/.
[19] Convolutional Neural Network UFLDL Tutorial Stanford University online: http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.
[20] Ciresan D Meier U Masci J Gambardella L and Schmidhuber 2011 High-performance neural networks for visual object classification Technical Report No. IDSIA-01-11 January 2011 Switzerland.
[21] Wang H Emmerich M and Plaat A 2018 Monte Carlo Q-learning for General Game Playing, arXiv:1802.05944.
[22] Chollet F 2017 Xception: Deep Learning with Depthwise Separable Convolutions, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 1800-1807.
[23] Transfer learning with a pretrained ConvNet, TensorFlow Tutorial online: https://www.tensorflow.org/tutorials/images/transfer_learning.
[24] Ciaburro G and Joshi P 2019 Python Machine Learning Cookbook: Over 100 recipes to progress from smart data analytics to deep learning using real-world datasets, 2nd Edition, Packt Publishing, ISBN: 1789808456.
[25] ImageNet online: http://www.image-net.org/.
[26] Nguyen L Lin D Lin Zh & Cao J 2018 Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation 2018 IEEE International Symposium on Circuits and Systems (ISCAS) DOI: 10.1109/ISCAS.2018.8351550.
[27] Zabir M Fazira N Ibrahim Z & Sabri N 2018 Evaluation of Pre-Trained Convolutional Neural Network Models for Object Recognition International Journal of Engineering and Technology. vol 7, pp 95-98, DOI: 10.14419/ijet.v7i3.15.17509.
[28] Varshney L Keskar N & Socher R Pretrained AI Models: Performativity, Mobility, andChange Preprint arXiv:1909.03290.
[29] NoxPlayer online: https://www.bignox.com/.