EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains

Daniel Jallov
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
daja@itu.dk

Sebastian Risi
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
sebr@itu.dk

Julian Togelius
Tandon School of Engineering
New York University
New York, NY, USA
julian@togelius.com

Abstract—**Neuroevolution (i.e. evolving artificial neural networks (ANNs) through evolutionary algorithms) has shown promise in evolving agents and robot controllers, which display complex behaviours and can adapt to their environments. These properties are also relevant to video games, since they can increase their longevity and replayability. However, the design of most current games precludes the use of any techniques which might yield unpredictable or even open-ended results. This article describes the game *EvoCommander*, with the goal to further demonstrate the potential of neuroevolution in games. In EvoCommander the player incrementally evolves an arsenal of ANN-controlled behaviors (e.g. ranged attack, flee, etc.) for a simple robot that has to battle other player and computer controlled robots. The game introduces the novel game mechanic of "brain switching", selecting which evolved neural network is active at any point during battle. Results from playtests indicate that brain switching is a promising new game mechanic, leading to players employing interesting different strategies when training their robots and when controlling them in battle.**

I. Introduction

While scripted AI can become predictable and allows players to find ways to exploit its limitations, neuroevolution (i.e. evolving artificial neural networks), allows agents to evolve and adapt in real time to the player [1, 2, 3, 4]. Game designers are however in general reluctant to include evolutionary algorithms or other learning algorithms for non-player character (NPC) control. One important reason is that the outcome of this process might be too unpredictable; the AI can end up acting "weird", breaking the gameplay experience for the player [1]. Furthermore, it is likely that most current games are designed so as to rely on predictable AI in the first place because game designers have little experience with learning AI algorithms. Thus an unfortunate cycle is perpetuated, in which learning AI are not used in games because games are designed to not need learning AI.

However, if games are specifically designed around the fact that evolution can be unpredictable (i.e. through making evolution part of the core game mechanics), evolutionary AI can create new and interesting challenges every time the game is played [1, 2, 3]. In addition to a constant stream of novel challenges, NPC behaviors would not need to be hand-crafted and painstakingly quality-controlled if automatically generated by an evolutionary algorithm.

The promise of this approach has been demonstrated in a variety of different games, in which neuroevolution (NE) algorithms allow both the NPC behaviors and the in-game content to adapt while the game is being played [4]. Two notable examples are Galactic Arms Race (GAR) [2], where NE generates new weapons as the player plays the game, and Neuroevolution of Robotic Operatives (NERO) [1], a game in which the player trains an army of robots controlled by ANNs for battle.

This paper introduces the game *EvoCommander*, which extends and combines ideas introduced by NERO and GAR, to further increase the player's engagement with the game. In EvoCommander the player trains a single robot for battle, by evolving an arsenal of neural controllers. The player can decide on a training regime for these controllers and evolve them incrementally and interactively. In contrast to NERO, in which the player takes the role of an observer after the training phase is completed, the novel game mechanic[1] introduced here allows players to *switch between brains* during battle as an indirect way of controlling their robot. Evolving brains displaying the "right" behavior, choosing a subset of evolved brains to take into battle, and deciding when to switch between them during the fight are the challenges that the players have to master in EvoCommander.

The results in this paper suggest that players perceive evolution and brain switching as a fun and engaging game mechanic. Players are able to evolve brains that solve fairly complex tasks and complete the provided game missions. When playing online against each other, players employ very different strategies (with varying degrees of success), indicating that brain switching is a useful game feature. As part of this project we also release UnityNEAT[2], a port of SharpNEAT[3] to Unity 4 and 5, with the hope to encourage more NE-based game projects. In the future, the novel game mechanic could be extended to a wide variety of other genres, thereby creating entirely new type of engaging games that would not be possible without advanced AI.

The next section provides relevant background and Sec-

[1]We use the term game mechanic in the sense defined by Sicart [5]
[2]Source code at https://github.com/lordjesus/UnityNEAT
[3]http://sharpneat.sourceforge.net/

tion III introduces the EvoCommander game. The playtests and experimental setup are described in Section IV, followed by the single- and multiplayer results in Sections V and VI. Implications of the results and future work are discussed in Section VII.

II. Background

This section reviews relevant work on NE, incremental evolution and NE-enabled games.

A. *Neuroevolution of Augmenting Topologies (NEAT)*

In neuroevolution (NE) [6, 7], which belongs to the class of stochastic search-based optimization methods, ANNs are trained through evolutionary algorithms. NE is especially useful in domains with no available training data or when the optimal network topology is not known in advance. NE searches for the best solution in a given domain, traditionally guided by a fitness function that measures the overall evolutionary progress. As with all evolutionary methods, the design of the fitness function is important in order to guide the evolution in the right direction.

The particular NE algorithm which we use to evolve the ANNs in EvoCommander is *Neuroevolution of Augmenting Topologies* (NEAT; [8]). NEAT is capable of evolving both the weights and the topology of an ANN at the same time. Additionally, NEAT allows effective search even on problematic fitness landscapes, through features such as speciation and innovation protection. Little tuning is usually required when applying NEAT to a new problem. In particular, NEAT does not require the user to decide on the topology of the ANN; only the number of input and output neurons have to be specified a priori.

NEAT starts out with a population of simple networks without any hidden nodes and only direct connections from the inputs to the outputs of the ANN. Hidden nodes and additional connections are then added to the network during evolution through mutations and cross-overs. The benefits of this approach are twofold. First, the evolved neural networks start minimal (i.e. no bloated networks), preventing NEAT from searching in unnecessarily high-dimensional search spaces. Additionally, complexification of networks over time leads to more and more advanced solutions, thereby creating complex behaviours evolved from simple foundations. This mechanism matches well with the idea behind EvoCommander, which lets players start with simple tasks and interactively evolve increasingly complex behaviours. Importantly for this paper, NEAT has also already been shown to perform well in a variety of game domains [9, 3, 2, 4].

B. *Incremental evolution and modular networks*

Evolution of complex behaviour can be difficult and sometimes impossible when starting from a random population, as evolution might get stuck in local optima in the search space. Therefore, players in EvoCommander can *incrementally* evolve complex robot behaviors.

Successful application of incremental evolution include the work by Gomez and Miikkulainen [10] in a predator-prey domain. By breaking up the task of preying into several, more simple sub-tasks, evolution was able to first find a solution to catch a stationary prey, then a slowly moving prey, and finally a fast moving prey. By incrementally evolving the predator on increasingly complex tasks, a solution was found to the original problem that was not solvable with direct evolution. The idea of incremental evolution has been shown to apply well to other domains, for example Togelius and Lucas [11] incrementally evolved simulated car racing controllers to drive well on a series of increasingly complex tracks.

Incremental evolution was also successfully applied to the complex task of helicopter control. De Nardi et al. [12] evolved a controller for a simulated helicopter to follow a path of given waypoints. While directly evolving a monolithic controller failed, breaking the task into multiple subtasks and evolving three separate neural networks that control different aspects of flying lead to a well-performing modular network architecture.

Incremental evolution and a modular network architecture were combined by Togelius [13] to evolve the layers in a subsumption architecture robot controller. Thompson and Levine [14] employed a similar method in the EvoTanks domain. In EvoTanks two tanks battle each other, which resembles the setup of EvoCommander. However, EvoTanks is a research testbed that does not include any form of player interaction. Schrum and Miikkulainen [15] evolved modular networks for Ms. Pac-Man, where different networks perform different tasks in gameplay, with good results. This echoes earlier results by Calabretta et al. [16] on robot control. Modularity is thought to play an important role in biological neural networks [17].

EvoCommander allows players to either start evolution from scratch or from an already existing brain, thus facilitating incremental evolution. In effect, players can create a whole family tree of networks. The game thus resemblances applications such as Picbeeder [18], in which users evolve images branching from existing images, or EndlessForms [19], in which users collaboratively evolve three-dimensional shapes.

C. *Neuroevolution in Games*

Neuroevolution algorithms have shown promise in a variety of different kinds of games, ranging from learning to control PacMan purely from raw visual input [20], or forming teams of PacMan ghosts trained through NEAT [21], over on-line adaptation [22, 23] and imitation learning in the TORCS car racing game [24, 25], to controlling bots in a first-person shooter through a hierarchical neural network controller [26]. There is also a long history of evolving neural networks to play classic board games such as Checkers [27]. An overview of successful applications of NE in games is given in Risi and Togelius [4]. Additionally, NE allows entirely new types of games and some games were designed specifically to showcase the possibilites NE affords game designers.

One such example is Galactic Arms Race (GAR; [2]), which is a two-dimensional space shooter with weapon particle systems that evolve while the game is being played (Figure 1,left).

Fig. 1: **Neuroevolution-based Games.** Neuroevolution enables innovations in a variety of different game genres. In Galactic Arms Race (left), the players can evolve an unlimited variety of ANN-encoded particular weapons (picture from Hastings et al. [2]). Petalz (middle) shows that NE can also generate content for social games. Players can interactively breed flowers and share them with each other through a global marketplace (picture from Risi et al. [3]). The objective in NERO (right) is to evolve a team of robots that can battle teams evolved by other players. NERO allows players to create increasingly complex behaviours enabled by evolutionary algorithms (picture from Stanley et al. [9]).

The idea in GAR is that the number of times a weapon is fired can serve as an indication of how much the player enjoys that particular weapon, allowing the game to adjust to preferences of the player. A player in GAR can hold three weapons at a time and sees a preview of new weapons before replacing his current ones. The idea of a limiting arsenal of evolved artifacts (e.g. weapons) that the player can chose from during battle is also one of the main features of EvoCommander. However, while the player in GAR can only indirectly influence the type of weapons that should evolve next, players in EvoCommander can train their tanks for a variety of specific and potentially complementary behaviors.

NE has not only shown potential in competitive games but also social video games. One such example is Petalz [3], a social Facebook game that allows players to interactively evolve flowers and sell them on a virtual marketplace (Figure 1,middle). Each flower is encoded by a special type of neural network and evolved by NEAT. The players interact with the flowers by either pollinating them (mutation), or cross-pollinating two flowers (cross-over), which results in flower seedlings that can be planted and grown.

The game most closely related to the approach taken in this paper is *NeuroEvolving Robotic Operatives* (NERO) [9]. In NERO the player trains a team of virtual robots for combat against other players' teams. To train the robots, the player can set up objectives through sliders in the UI to reward the robots to perform certain behaviours. These objectives can be approaching or fleeing from a target, attacking a target or avoiding enemy fire. The player can interact with the evolution by changing the fitness objectives or by acting as an enemy himself (Figure 1,right). The goal of the training is to create a team of robots that can handle themselves in battle. Once the training is complete, the team can compete against other players online.

In this paper, the idea of training robots through evolution is inspired from NERO. However, there are several important design differences between EvoCommander and NERO, most importantly the brain switching mechanic. In NERO the game "ends" when the training is done; during battle the player is reduced to a spectator role that now has no further control over the battle's outcome. EvoCommander aims to keep the players engaged during the battles by allowing them to control several aspects of the match, most importantly which evolved behavior is active at any point during the fight. The hope is that this approach should offer a more satisfying playing experience and engage the player *both* before and during the battle.

III. EvoCommander

This section presents the game *EvoCommander*[4], its mechanics and details its development. The goal of the game is to evolve behaviors that can control a simple tank in a battle against another human or computer-controlled robot. The player can decide which types of behavior to evolve, ranging from tasks such as moving around, attacking with melee weapons, attacking with ranged weapons and shooting mortars. After evolving an arsenal of ANN-controlled behaviors, players can test their robots by completing missions (single player) or by competing against each other online.

In effect, the game offers three distinct challenges: First, the player has to decide what behaviors to evolve and design their training regimen. Second, the player has to chose a subset of the evolved behaviors (a total of four) to take into battle; the goal is to chose behaviors that work well together, such as an defensive and an offensive one. Third, during battle the player has to decide which behavior to use at what point during the fight. The hope is that these three mechanisms open up tactical perspectives not possible with only a single ANN controlling the robot, engaging the players even after the training period is completed. Thus, EvoCommander offers the players a variety of different areas to master, which is an essential aspect of many successful competitive video games.

The game consists of two main parts: a **training part**, in which the player evolves the behaviors for the robot, and

[4]The game can be downloaded from http://jallov.com/thesis/. A video showing the basic game mechanics can be found here: http://youtu.be/Fz-R4xwtsGM

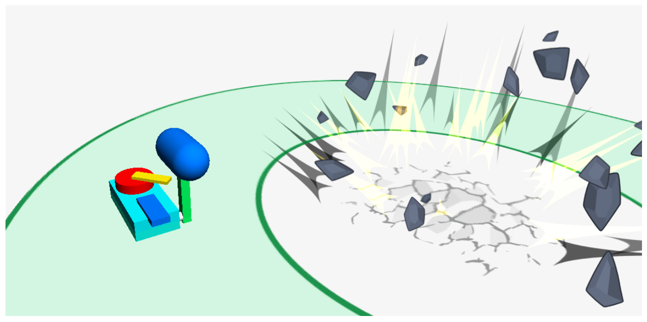


Fig. 2: **EvoCommander Robot.** This figure shows the robot standing in the impact of its own mortar explosion. The robot tank is able to move forward and backwards and turn in place. It has three weapon slots for melee, ranged and mortar weapons. In EvoCommander the player does not control the robot directly but instead decides which of four evolved brains should be active at what time during battle.

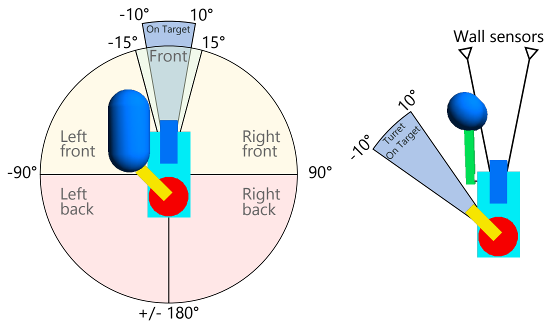


Fig. 3: **Robot Sensors.** The robot in EvoCommander has seven different sensor types. It can detect enemies with its pie-slice sensors that activate when a line from the target to the centre of the robot falls within the pie-slice. The two wall sensors indicate the closest distance to the closest object in that direction. The other inputs are a on-target sensor, distance to target, turret-on-target sensor and nearest pickup sensor.

a **battle part**, in which the player can switch between the evolved behaviors to compete against other robots. Before detailing these two main components, the EvoCommander robot together with its controller architecture is described next.

A. Robot and Neural Network Setup

The robot is a small, tank-like vehicle that is able to move forward and backwards and turn in place (Figure 2). While the robot attacks, its speed is reduced and it is then regained at a constant rate after the attack. (This adds a small movement penalty to all attacks.) The robot starts with 100 health; when reaching a level below zero it explodes and the player loses the battle. The tank has three slots for different types of weapons that the player can equip before the training phase begins:

1) **Melee Weapon:** The robot has a slot for one melee attack weapon. If opponents are hit with the melee weapon, they lose health according to the damage of the currently assigned weapon. Whenever the robot attacks, it incurs a speed penalty. Therefore it is not beneficial for a robot to constantly use its melee weapon, since it will heavily impede its movement speed.
2) **Ranged Weapon:** In front of the robot is a slot for its ranged weapon. This weapon has a range of 50 meters and instantly hits. The ranged weapons generally fire much quicker than the melee weapons, but deal substantially less damage and result in lower speed penalties.
3) **Mortar Weapon:** On top of the robot is an independently turning turret. This turret fires mortars, which are big cannon balls that inflict area damage on hit. The damage falls off linearly from the center of impact and out to the periphery of the impact area. Firing mortars results in a 80 % - 100 % speed penalty, rendering the robot almost completely stationary while firing.

The different weapons in the game allow the players to tailor the robot to their particular playing style and to choose what suits them best. For each weapon category three weapons can

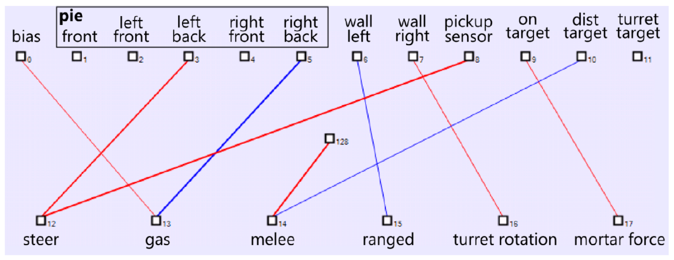


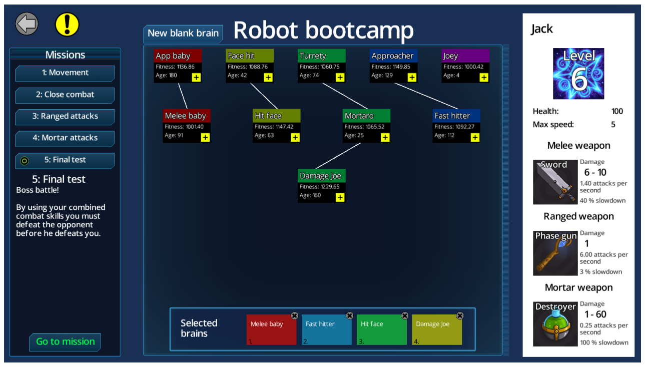
Fig. 4: **Evolved Network Example.** A network evolved for 50 generations that can approach a target. Red connections are positive weights, blue connections are negative weights. The width of a connection is proportional to the value of the weight, i.e. thicker lines means larger numerical values. NEAT is able to discover a compact ANN that is able to solve the task defined by the player.

be selected. The weapons in each category have almost equal damage per second, but vary in attack speed and how much they slow down the robot's movement speed.

The robot is equipped with seven different sensor types in order to perceive the world around it (Figure 3). There are 11 active input sensors: five pie slices, two wall sensors, an on-target sensor, a distance to target sensor, a turret-on-target sensor and a distance to nearest pickup. These inputs are fed to a neural network evolved by NEAT.

The 360° around the robot are divided into five pie slices. If the target is in pie slice p_i (within 100 meter), the corresponding input is activated: $p_i = 1 - \frac{d}{SR}$, where d is the distance to the target and SR is the sensor range, set to 100 meters. The input is scaled between [0, 1], with a higher activation the closer the target is to the robot.

The on-target sensor is activated if the target is within (-10°, 10°) in front of the robot and the turret-on-target sensor is activated if the target is in front of the turret. The activation level t is calculated as follow: $t = 1 - \left|\frac{\alpha}{10}\right|$, where α is the angle between the forward vector and the vector from the robot



5: Final test
Boss battle!

By using your combined combat skills you must defeat the opponent before he defeats you.

App baby — Fitness: 1136.86, Age: 180
Face hit — Fitness: 1088.76, Age: 42
Turrety — Fitness: 1060.75, Age: 74
Approacher — Fitness: 1149.85, Age: 129
Joey — Fitness: 1000.42, Age: 4
Melee baby — Fitness: 1001.40, Age: 91
Hit face — Fitness: 1147.42, Age: 63
Mortaro — Fitness: 1065.52, Age: 25
Fast hitter — Fitness: 1092.27, Age: 112
Damage Joe — Fitness: 1229.65, Age: 160

Selected brains: 1. Melee baby, 2. Fast hitter, 3. Hit face, 4. Damage Joe

Jack
Level 6
Health: 100
Max speed: 5

Melee weapon: Sword — Damage 6 - 10, 1.40 attacks per second, 40 % slowdown
Ranged weapon: Phase gun — Damage 1, 6.00 attacks per second, 3 % slowdown
Mortar weapon: Destroyer — Damage 1 - 60, 0.25 attacks per second, 100 % slowdown

Fig. 5: **Robot Bootcamp.** The bootcamp allows the player to create and choose robot brains, select unlocked weapons and perform the missions. Players can chose to either create new brains from scratch or elaborate on previously evolved ones thus creating a whole hierarchy of networks.

to the target.

Two wall sensors originate from the center of the robot and detect any walls to its left or right side. If the sensors detect a wall (up to a maximum distance of 100 meters) the network inputs are activated proportionally to the distance between the robot and the wall (normalized between [0, 1]).

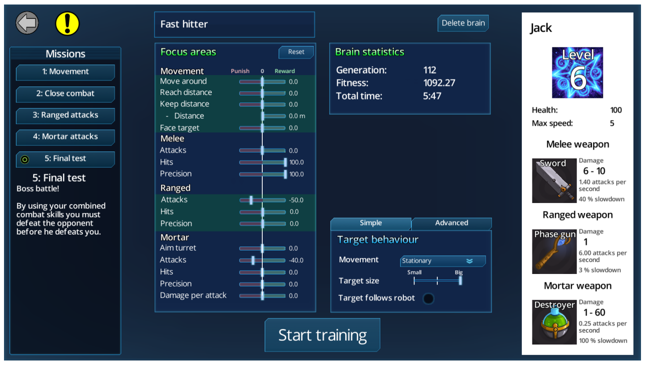
The six outputs of the network are *Steer*, which rotates the robot in place, *Turret rotation*, which turns the turret, *Gas*, for forward/backward moving, *Melee attack* and *Ranged attack* which activate melee/ranged attacks if the output value is greater than 0.5, and *Mortar force*, which determines the length of the mortar shot and fires if the output value is greater than 0.5. All output values are scaled between [-1, 1].

The aforementioned inputs and outputs allow players to evolve a diversity of different robot behaviors. One such example network evolved by NEAT is shown in Figure 4, which is able to approach a target in the game.

B. Training Mode

The main overview screen in EvoCommander is the *Robot Bootcamp* (Figure 5). Here the player can train the robot, see its progress and the mission progress. For each mission players can find information on what is required to complete them and chose which missions to play. Additionally, the player can switch weapons for each of the three weapon slots. To gradually familiarize the players with the game, more missions and weapons become available as they proceed (e.g. at Level 1 the robot is only able to move around the arena without using any of its weapons). Furthermore an important part of this game is an extensive tutorial system, which is activated when the player first starts the game and which can be opened again later if needed.

To train their robots, players can create and select brains for training and battle from the middle panel in the bootcamp screen. Players can either decide to start from a new blank brain or branch from an already existing one. After selecting a brain, the player is taken to the *Training Area* (Figure 6).



Fast hitter

Movement
Move around 0.0
Reach distance 0.0
Keep distance 0.0
- Distance 0.0 m
Face target 0.0
Melee
Attacks 0.0
Hits 100.0
Precision 100.0
Ranged
Attacks -50.0
Hits 0.0
Precision 0.0
Mortar
Aim turret 0.0
Attacks -40.0
Hits 0.0
Precision 0.0
Damage per attack 0.0

Generation: 112
Fitness: 1092.27
Total time: 5:47

Movement: Stationary
Target size
Target follows robot

5: Final test
Boss battle!

By using your combined combat skills you must defeat the opponent before he defeats you.

Jack
Level 6
Health: 100
Max speed: 5

Melee weapon: Sword — Damage 6 - 10, 1.40 attacks per second, 40 % slowdown
Ranged weapon: Phase gun — Damage 1, 6.00 attacks per second, 3 % slowdown
Mortar weapon: Destroyer — Damage 1 - 60, 0.25 attacks per second, 100 % slowdown

Fig. 6: **Training Area.** In the training area the player can choose which areas to focus on during the evolutionary training and how the practice target should behave. Information on how many generations the network was already evolved for is also given. The sliders in the UI allow the players to precisely describe a variety of different robot behaviours, thereby giving them the ability to evolve an arsenal of behaviours that work well together in battle.

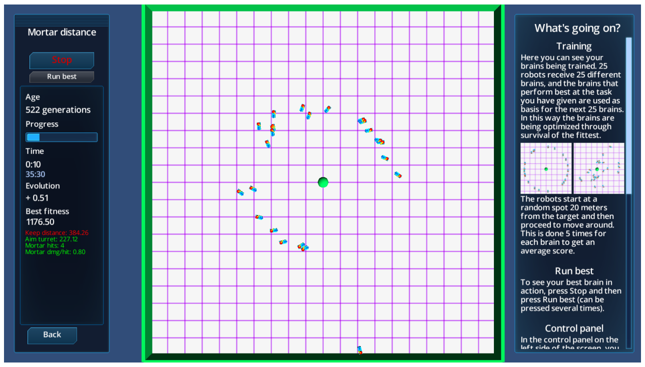
1) Setting Up Training:

1) Setting Up Training: In typical scenarios involving evolutionary algorithms, the user has to define a fitness function a priori that guides the evolutionary search. The approach in EvoCommander is more interactive and allows players to describe ideal robot behaviors by adjusting sliders in the UI (Figure 6), resembling the setup in the NERO game [9]. For example, the sliders specify how much the robot is rewarded or punished for staying close to its opponent, facing its opponent, hitting the target with a melee weapon, etc. The sliders are grouped in semantic categories, so called *focus areas*:

1) **Movement:** The robot is rewarded or punished for moving around, for staying close to the target, and for facing the target.
2) **Melee:** The robot is rewarded or punished for melee attacks, for hitting the target, and for precision (hitting on each attack).
3) **Ranged:** The robot is rewarded or punished for ranged attacks, for hitting the target, and for precision.
4) **Mortar:** The robot is rewarded or punished for mortar attacks, for hitting the target, for precision, and for dealing as much damage as possible per hit (damage falls off linearly from the center of impact).

By adjusting the sliders for each of the focus areas the player has detailed control over the robot's evolutionary training. Each focus area is directly related to a component in the fitness function, which is a weighted linear combination of all the focus areas: $f = b + \sum_{i \in F} a_i \cdot w_i$. Here F is the set of active focus areas (some focus areas are disabled if the player has not yet reached a certain level), a_i is the observed value of the focus area scaled to [0, 1], w_i is the weight given by the player through the UI sliders in the range [-100, 100] and $b = 1000$ is a bias to prevent negative fitness values.

Because players new to EvoCommander might be overwhelmed by the variety of different settings, they can also



Mortar distance

Age
522 generations
Progress
Time
0:10
35:30
Evolution
+ 0.51
Best fitness
1176.50
Keep distance: 384.26
Aim turret: 227.12
Mortar hits: 4
Mortar dmg/hit: 0.80

What's going on?

Training

Here you can see your brains being trained. 25 robots receive 25 different brains, and the brains that perform best at the task you have given are used as basis for the next 25 brains. In this way the brains are being optimized through survival of the fittest.

The robots start at a random spot 20 meters from the target and then proceed to move around. This is done 5 times for each brain to get an average score.

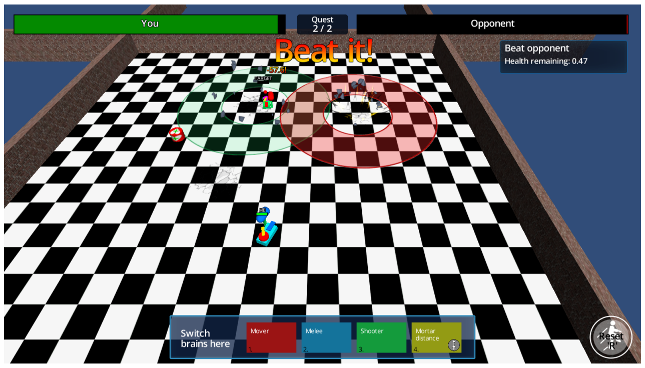
Run best

To see your best brain in action, press Stop and then press Run best (can be pressed several times).

Control panel

In the control panel on the left side of the screen, you

Fig. 7: **Optimization Screen.** From the optimization screen the player can observe the evolutionary process in action. 25 individuals are concurrently evolved and their behaviours displayed to the player. The current generation number and best overall fitness is also shown. At any point the player can choose to run the best individual evolved so far or stop the evolutionary search.



Beat opponent
Health remaining: 0.47

Fig. 8: **Mission Screen.** This figure shows an example of a battle between the player and the computer-controlled robot during EvoCommander's missions. The player is about to win the battle, having brought the opponent down to 0.47 health and is just in need of one more hit to win. The four brains chosen by the player are shown at the bottom, with the fourth brain 'Mortar distance' selected as currently active.

chose from a number of predefined target behaviors (e.g. stand still, move around or follow the target) with already specified slider settings.

Players can also select how the practice target should behave. There are four different movement patterns: Target stands still, target moves back and forth, and target moves randomly. To allow the evolution of more complex behaviours, the target can also be set to follow the robot or its size can be adjusted through a slider in the UI. While it is easier to hit a large target, more precise firing often requires training on small targets. The player can see statistics about the evolved networks, such as the number of training generations and their fitness values.

2) Evolution in Action: When the player has chosen the ANN's focus areas the evolutionary training can be started (Figure 7). The *Optimization Screen* shows a square arena with a target in the middle, and 25 robots spawning at random positions in a 20 meter distance from that target. Each generation is evolved concurrently, with all the 25 robots being evaluated at the same time. For each generation each robot is run for five iterations of 20 seconds each, to evaluate how well the underlying ANN performs under varying starting positions. The game runs at an increased speed during this phase in order to complete as many generations possible in the shortest amount of time. The robots do not interact with each other in any way during training; the reason why parallel evaluation is used at at all is that the Unity game engine limits how much the simulation can be sped up. The fastest computers may be able to run the simulation at 25 times normal speed, while older computers only reach 12-15 times normal speed. Thus a generation in EvoCommander can be processed in around four to ten seconds.

After a generation is complete, the best individuals are kept for reproduction while the worst individuals are removed from the population. It usually takes about three to five minutes for evolution to find a brain that performs well on the task defined by the chosen focus areas (e.g. moving close to the target, shooting lasers at the target, etc.).

The left sidebar of the optimisation screen shows the number of generations passed, by how much the average fitness has increased (or decreased) and the fitness of the best individual evolved so far. The player is also presented with a detailed breakdown of how the best individual performed on the selected focus areas. The right sidebar gives an overview of how the system functions and describes its settings.

C. Battle Mode

1) Missions: At any point in the game the player can test the performance of the evolved behaviours in the missions (Figure 8). The missions are designed to give a gentle introduction to how the battles in EvoCommander work, preparing the player for fights against other players in the online arena. They consist of several quests each, which are usually structured in a way that the first couple of quests test the abilities of the newly unlocked weapons. Once the player completes a mission by beating the computer-controlled robot, the next level of weapons and missions are unlocked in the bootcamp. The last mission is a "Boss battle", in which the player has to beat an opponent with 150 health, opposed to the 100 health of enemies in the missions and the player's own robot.

Before going into battle the player chooses four of the evolved brains to take into combat. Part of the challenge is to choose the "right" combination of brains that will allow for an effective strategy. During battle the players can, at any point, switch between the brains in their arsenal, thereby activating a different behavior. Thus players in EvoCommander are free to decide to e.g. evolve one very general behaviour that is active for most of the battle, or evolve a variety of very specialised behaviours that they often switch between.

2) Champion's Arena: Once all the missions are completed, the player is encouraged to go to the *Champion's Arena* to fight against other players online. Any player regardless of level can enter the Champion's Arena, but players who are below level 4 will face a warning before entering; their robots are not yet fully developed and lack the offensive abilities of opponents of level 4 and higher. The arena resembles the robot bootcamp with the exception of the missions panel that has been replaced with an online lobby. The online lobby allows players to join open games or create new online games.

Matches against other players can either be played as a single game, a best out of three or a best out of five. The controls are the same as in the missions: The robot's behaviors is determined by the currently active brain, and players can at any point switch between the four brains they have chosen to take into battle.

D. EvoCommander's Beginner's Guide

An important aspect of a game build around novel AI techniques is to introduce players gradually to the main game features and guide them through potentially challenging aspects. Therefore EvoCommander includes a number of such features designed to improve the overall player experience, which were found useful through earlier play tests.

1) Level system: A level system gradually introduces the player to the complexity of the game by unlocking new features for each new level. At levels 2–4 new attack types are unlocked, and by reaching the last level the player should have all the skills needed to battle online.

2) Missions: The missions serve as an introduction to the game and the control system, and guide the player towards evolving behaviors that are useful in battle situations. The missions act as an indicator of whether the player has evolved brains that are "good enough" for online play. Each mission is designed to test the newly unlocked feature (e.g. get close to a target, flee from a target). When one mission is completed a new mission and new feature are unlocked. The five missions test the player in basic movement, the different attack types and concludes with a boss battle in mission 5. The UI and control scheme is the same in the missions as in the online arena, thus preparing the players for online battle.

3) Tutorials and hints: Because players are typically not used to game mechanics based on learning AI, EvoCommander can be challenging to master initially. There are a number of concepts the player has to be introduced to in order to convey a meaningful playing experience. Therefore an important part of this game is the many tutorials and hints that guide the player; almost every screen includes a set of tutorial dialogues that gives further explanations on how to play the game.

IV. Playtests

Single- and multiplayer play tests were conducted to investigate how the game was played and to test whether brain switching offers an interesting new game mechanic. In the single-player tests, data was recorded while the players were playing the EvoCommander missions. These tests also included some testing of the multiplayer functionality. In the main multiplayer testing, the behaviors of players competing against each other was analyzed. Information about brain switches during the matches was recorded, including how long each brain was active, and the amount of damage it inflicted. An interesting question in this context is how players use the mechanic of brain switching and if this system allows them to employ a variety of effective strategies. Additionally, a motivating aspect of many popular games is to find strategies that work well together. Therefore, the focus areas chosen by the players were also recorded to determine if EvoCommander's interface allows players to evolve complementary behaviors. Note that these studies do not test the computational efficiency of the evolutionary algorithms, mainly because the game is based on the well-known NEAT algorithm [8] and paradigm of input/output representation.

A. Experimental Setup

To use NEAT in the Unity game engine, the C♯ implementation of NEAT, SharpNEAT[5], was ported to work with Unity's Mono version to interact with the scenes in the game engine. This port is called UnityNEAT and is publicly available[6]. NEAT is run with a population size of 25. Sexual offspring (50%) did not undergo mutation. Asexual offspring (50%) had a 88.8% chance of weight mutation, a 0.5% chance of adding a node, and a 5% probability of adding a connection. The activation function in the ANNs is the following steepened sigmoid function: $S(x) = \frac{1}{1+e^{-4.9 \cdot x}}$.

V. Single-player Results

During our single-player test, a total of 62 players downloaded the game. Of these 62 players, 48 evolved behaviors (referred to as 'active' players). More than half of the active players completed the first mission, about 30 % unlocked all weapons by reaching level 4 and 17 % completed the entire game.

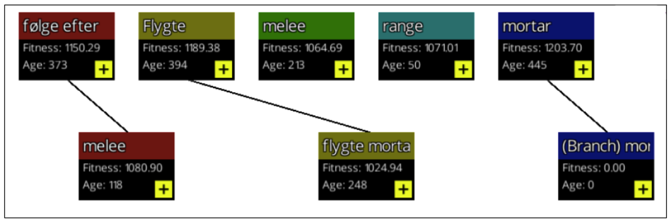
A. Neural Network Data

The players evolved a total of 557 ANNs, from which 165 are unique. The remaining 392 are archived versions of brains, which are automatically created when a player changes a brain's focus areas. Generally, as the players progress to higher levels, they create more ANNs and refine their existing ones, resulting in more versions of the same networks.

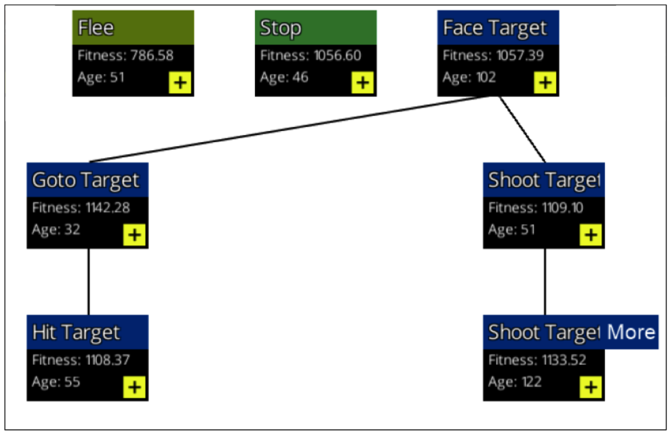
To get a better understanding of which focus areas are commonly used together, the focus areas of all unique 165 ANNs were mined with the Apriori association mining algorithm [28]. The Apriori algorithm can find all the rules in the form of $X \rightarrow Y$ (e.g. if somebody bought bread they also bought milk), with a certain support (probability that a transaction contains $X \cup Y$) and confidence (probability that transaction having X also contains Y). Five rules were found with a minimum support of 10% and confidence of 100%:

[5]http://sharpneat.sourceforge.net/

[6]https://github.com/lordjesus/UnityNEAT



(a)



(b)

Fig. 9: **Example Training Regimen.** While some players prefer to create new behaviors from scratch (a), other players prefer to first evolve networks with rudimentary skills (facing the target), and then incrementally elaborate on them to perform more complex tasks (b).

1) Ranged precision → Ranged hits (12% support)
2) Keep distance, Melee hits → Melee precision (10% support)
3) Move around, Melee precision → Melee hits (10% support)
4) Face target, Keep distance, Melee hits → Melee precision (10% support)
5) Keep distance, Melee attacks, Melee hits → Melee precision (10% support)

These rules suggest that players tend to use focus areas that go well together and that they understand the principles of how to train their robot. For example, training for *precision* together with rewarding *hitting* a target is a good strategy to encourage behaviors that hit the target often (i.e. *precision* rewards the robot for not doing unnecessary attacks while *hit* rewards the robot for hitting the target many times).

B. Training Battle Skills

EvoCommander allows players to incrementally train robot behaviors by changing what behaviors to reward as training progresses. Figure 9 shows two examples of how different players that finished the game approach the training in EvoCommander. While some players create most of their behaviors starting from scratch (Figure 9a), other players chose a single master brain to elaborate on. Figure 9b shows an example of a player that first evolves a brain that is able to face the target and then further evolves this brain to perform melee attacking and ranged shooting; both behaviors which rely on the ability to face the target.

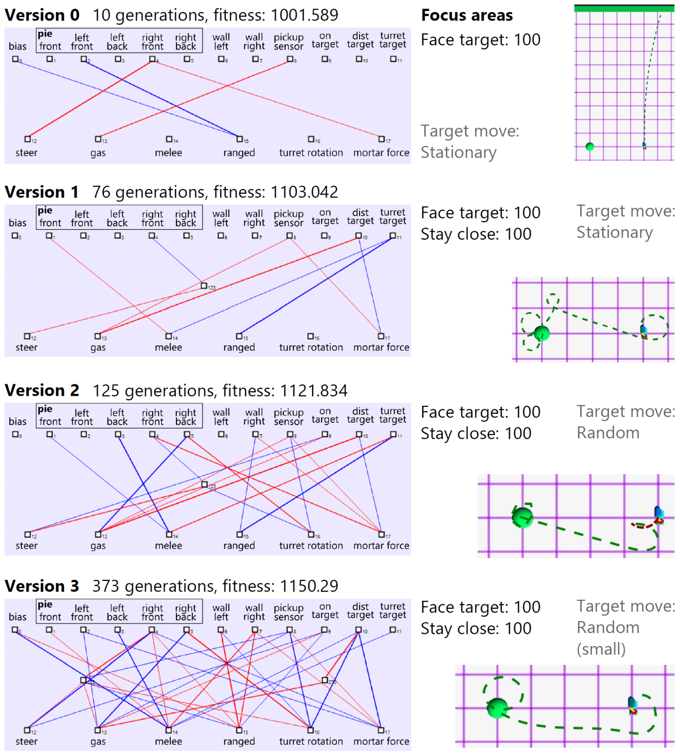
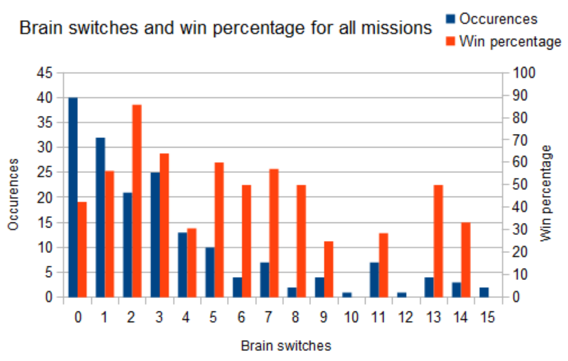


Fig. 10: **Incremental Evolution Example.** This figure shows the evolution of a behavior that is able to approach a randomly moving target. The main result is that players in EvoCommander can guide the evolutionary search towards relevant behaviors by incrementally and interactively influencing the evolutionary search.

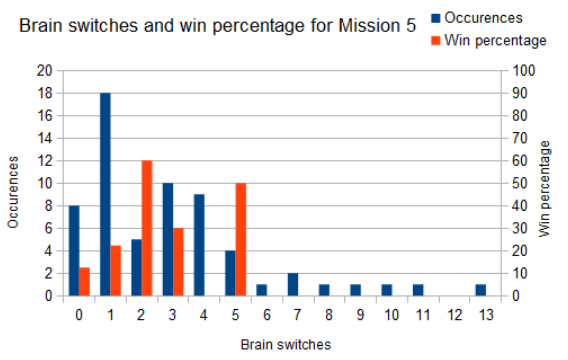
Figure 10 shows a breakdown of the training process of the brain, which can approach a moving target. Initially, the network is only trained to face a stationary target. After ten generations the player changes focus areas to both face the target and stay close to it. After 76 generations the network has increased in complexity and the robot is now able to approach the stationary target but uses quite a lot of turns in its approach. To further fine-tune the network the player decides to train against a moving target. This training proceeded until generation 125, resulting in a very efficient approaching behavior; once the robot is on target it goes straight for it. Finally, the player changes the size of the target to small and runs evolution for another 250 generations. This evolved network is even quicker at finding the target and proceeds straight to it once it is found.

C. Brain Switches

The number of brain switches and outcomes for each mission played is shown in Figure 11a. There is a clear tendency towards using few switches in the missions; in two thirds of the missions the players only switched brains between



(a) All Missions



(b) Boss Battle

Fig. 11: **Brain Switches** Histogram of brain switches and percentages of matches won for (a) all missions and only for the boss battle (b). The main result is the number of brain switches is relatively low in the missions but that only relying on one evolved behavior is not a winning strategie.

zero and three times. However, relying on only one evolved behaviors seems to not be very efficient, as it only wins 42,5% of the times. Two brain switches on the other hand is much more efficient and more than doubles the chances of success, yielding a win percentage of 85,7%.

It is interesting to analyze how the use of brain switches changes in different situations in the game. For example, while the player has to fulfill certain goals in the missions (e.g. approach the target), the only goal for the player in the boss battles is to beat their opponent. This difference is also reflected in the number of brain switches (Figure 11b). In the boss battles, one brain switch is the most common, but not the most efficient strategy with only 22.2% wins. More switches seems to be a better choice for the boss battle, with two switches reaching the highest win rate of 60%. These result suggest that the number of switches does in fact influence the robot's success rate and that the ideal number depends on the current goals in the game.

VI. Multiplayer Results

A second playtest was performed on site at the IT University of Copenhagen with a specific focus on the multiplayer aspect

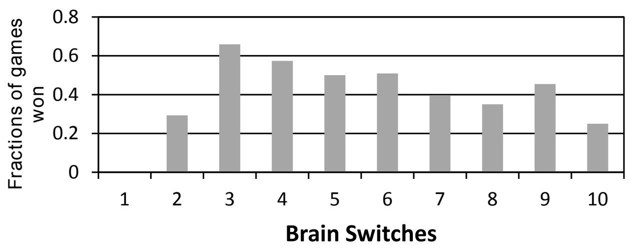


Fig. 12: **Brain Switches During the Multiplayer Playtest.** Switching brains three times during a match significantly increases the chances of winning compared to two switches. However, switching two often actually reduces the chances of winning.

TABLE I: Multiplayer Playtest Results

Total Number Players	20
Total Matches	143
Average Matches Per Player	34 (sd=29)
Average Match Length	40s (sd=19)
Brains Evolved	69
Average Generations Per Brain	120 (sd=111)
Max Generation	552
Average Brains Used in Battle	2.72 (sd=0.86)
Total Switches	4,858
Mode Switches	4
Max Switches	102
Min Switches	2
Total Melee Attacks	8,137
Total Ranged Attacks	38,605
Total Mortar Attacks	3,865

of the game. A total of 20 testers played 143 games; Table I summarizes the results. Brain switching is shown to be a frequently used game mechanic, with a total amount of 4,858 switches. Every player switched at least twice per match, and the maximum number of switches per player was 102. The mode of brain switches was four. Additionally, with 120 generations on average players invest considerable time in evolving their brains.

Importantly, the number of brain switches also influences the outcome of the match (Figure 12). While switching two times only results in winning the game 29% of the time, switching three times during a match seems to give the best chances with a percentage of 67%. This difference is significant, following Fischer's exact test ($p < 0.05$). Switching more seems to generally result in decreased win chances. Interestingly there is a significant positive correlation between the player's health and the number of switches ($r = 0.69$, $p < 0.05$), i.e. the higher the player's health, the higher the number of brain switches.

A. Questionnaire results

In order to analyze the players' subjective experience, we asked them to fill out a questionnaire after they played the game. A total of 14 questionnaires was filled out. To characterize patterns in the players' responses we labeled them with tags and then created aggregated tags consisting of several related tasks.

- **Most interesting part:** Multiplayer (4), brain switching (2), training/evolution (7)
- **Least interesting part:** Long training time (4), training too overwhelming/too many parameters (7)
- **What could have been improved:** training could be made clearer (4), ability to evolve more complex and different behaviors (2)
- **Did you identify any strategy for playing (and winning):** yes (10), switching brains/switching at the right time (8), approaching target (with laser) then melee (3), no (4)
- **Other comments:** fun (6), cool (2), fresh concept (1), boring (1), frustrating (2), physic bugs (1), lack of particle effects (1)

The results suggest that the players found the aspect of battling other players in the online arena especially engaging and that the combination of brain switching and evolution works well together. Even though the players enjoyed the game in general and many reported they had fun, a recurring theme was also that the training is too complex and takes a rather long time.

Importantly, players reported that they found a variety of strategies for playing and winning in multiplayer mode. Some examples include: (1) approach target (with laser) then melee, (2) approach, shoot, retreat, and (3) four brain strategy: mortar over distance, approach/melee, one to counter melee attacks, one to stay still to break out of loops.

Players playing against each other online employed very different strategies due to the differences in the behaviors they evolve. Some players preferred only using a few effective behaviors, whereas others used a combination of networks to control their robot more closely. The next sections presents a breakdown of four online battles that are representative of different strategies that players in EvoCommander developed.

B. Multiplayer Battle Examples

Figure 13a shows a progression of two mortar battles from the same match. Each player uses only two brains throughout the battles; Player 1 chooses 'Damage Joe', which is a mortar firing brain, and 'Hit face', which is a brain that is good at firing the ranged weapon over long distances. Player 2 chooses the brains 'følge efter' ("pursue"), which is good at approaching and following the opponent, and 'ny mortar' ("new mortar"), which is a mortar firing brain. In the first battle, Player 1 quickly switches to a brain firing ranged weapons ("Hit face") and Player 2 employs and approaching brain. After a few seconds Player 2 is close enough to Player 1 to switch to the brain "ny mortar" and begins hitting Player 1 with mortars. While Player 1 inflicts some damage with his ranged weapons, Player 2 deals much more damage with the mortar and wins the game comfortably. In the second battle, Player 1 resolves to only employing the "Damage Joe" brain throughout the entire battle. Player 2 first employs a brain to approach until the player is in range, and then also switches to the brain that uses mortar attacks.

Figure 13b shows an example of a more melee oriented match. In the first battle, Player 1 mostly just uses a brain that fires ranged weapons and at the same time employs the melee weapon ("Hit face") and Player 2 is trying to get close to Player 1 with an approaching brain. After receiving 15 ranged damage, Player 2 switches to his own ranged brain, but fails to hit Player 1 with any ranged shots. Once the players get close enough, Player 2 switches to the melee brain. After around 19 seconds Player 2 tries to once again face the opponent with an approaching brain but is destroyed by Player 1 in the process. In the second melee battle both players rely on their ranged and approaching brains to get close to each others. Once they are close, they employ two different strategies. Player 1 uses only his "Hit face" brain, which both inflicts melee damage and ranged damage at the same time. Player 2 quickly switches between a melee and approaching brain. The melee brain is good at inflicting damage, but it also causes the robot to rotate, which can be countered by switching to the approaching brain.

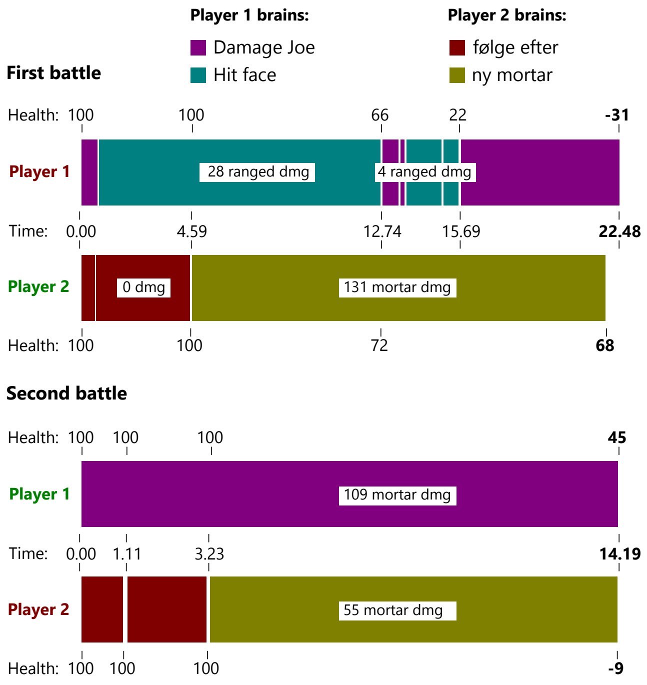
The battle examples and the questionnaire results demonstrate that EvoCommander allows players to design unique strategies with different tactical implications by (1) deciding what behaviors to evolve, (2) choosing which subset of the evolved behaviors to take into battle, (3), deciding which behavior to use at what point during the fight. The next section discusses some implications of this novel game mechanic and points out future work.

VII. Discussion and Future Work

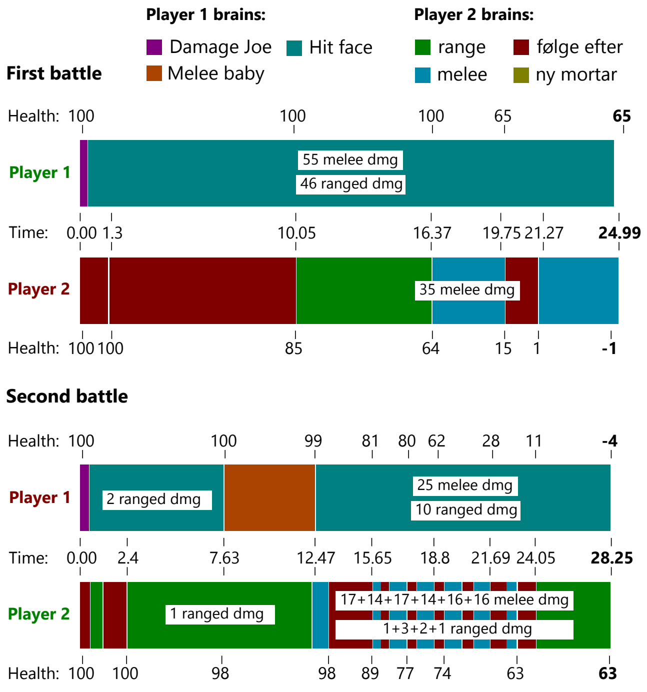
More than half of the players who played the game managed to complete the first level, and one out of six completed the entire single player campaign. More than 550 brains evolved by 62 players suggest that the game is engaging enough for the players to keep refining their existing brains and create new ones. The novel game mechanic of brain switching allows players to employ a variety of different strategies and separates this game from NERO [9]. Players are not merely spectators during the battles, but actively influence their outcomes.

The analysis of the number of brain switches in the single player campaigns shows that the players mostly preferred to use none or one brain switch. This might however be a result of the mission's level design. Each mission only has a single objective, which in many cases could be solved by a single brain (e.g. hit the opponent with the hammer or with lasers). The design of the missions were intended to encourage the player to evolve an arsenal of complementary brains that would make an all-round robot for later combats against other players (e.g. one brain for approaching the opponent, one for for shooting either lasers or mortars, and another one for dealing melee damage once the robots got in close combat). And in fact, players that did use more brain switches (two) in the missions and boss battles had a higher chance of winning, which is likely related to a proper use of these allround skills.

The qualitative and quantitative results collected during the multiplayer playtest show that players employ a variety



(a) Mortar Battles



(b) Melee Battles

Fig. 13: **Example Battle Timelines.**

of different tactics when playing against each other. Some players had a preference of using very few behaviors and brain switches, while other players employed more behaviors and switched rapidly between them. This strategic richness is a direct consequence of the brain switching mechanic.

One of the challenges in creating a game based on advanced AI techniques is to create a player-friendly wrapper for the evolutionary component of the game. Players are not used to terminology such as 'Fitness function' and 'Evolutionary algorithm', and might not directly know what a 'generation' or 'iteration' means in the context of a game. These concepts have to be presented in a way that is more intuitive for the player to grasp and this challenge is not fully solved in EvoCommander, as also suggested by some of the answers in the questionnaire. In order to appeal to more players, the process of setting up the training parameters should be less complex or be introduced by a step-by-step tutorial.

The sheer time required to perform evolution is also a factor to consider and people noted that this part could have been improved. Currently, it usually takes about three to five minutes to get an approaching brain working starting from scratch – given that the parameters are set correctly. Therefore, an important area for future research is to increase the chances of finding useful behaviors and to increase the speed of the evolutionary search in general. Evolution could for example be run in the background while the player plays the game. Additionally, instead of starting from random populations, the initial populations could be seeded with pre-evolved behaviors.

Some planned features still need to be improved and were not included in the current version of the game. Health pickup sensors are already a part of the robot's ANN, but the input to the sensor is not yet implemented. Such an addition would allow more tactical opportunities, such as players training their robot to flee when low on health and seeking health pickups. Additionally, in the future the robot could be trained against other evolved robots instead of the current dummy target. This way, the training would bear more resemblance to the actual multiplayer battles. Players could also be allowed to trade robot brains with others or offer them on a global marketplace, similar to the flower marketplace in Petalz [3].

VIII. Conclusion

A new video game called EvoCommander was introduced, which consists of a training part where evolution is used to train a robot to achieve player-defined goals, and a battle part where players take their robots online and battle against other players' robots. The novel game mechanic introduced in this game is allowing the player to switch the brains of the robot during battle, thus giving the player indirect control over the robot. An analysis of the evolved brains showed that players often use focus areas together which are semantically related to the task and increase the effectiveness of the robot. These results indicate that evolution works as an interesting game mechanic, and that players are able to evolve diverse behaviors. Additionally, the results show that EvoCommander allows players to employ a variety of different strategies thereby providing room for continuous improvement and innovation in competitive gaming. The main result is that brain switching works as a game mechanic. It could potentially be extended to many other game genres in the future.

Acknowledgments

Special thanks to the playtesters of EvoCommander.

References

[1] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 653–668, 2005.

[2] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 4, pp. 245–263, 2009.

[3] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley, "Combining search-based procedural content generation and social gaming in the petalz video game." in *AIIDE*, 2012.

[4] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[5] M. Sicart, "Defining game mechanics," *Game Studies*, vol. 8, no. 2, pp. 1–14, 2008.

[6] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[7] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/?stanley:ec02

[9] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving neural network agents in the nero video game," *Proceedings of the IEEE*, pp. 182–189, 2005.

[10] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, vol. 5, no. 3-4, pp. 317–342, 1997.

[11] J. Togelius and S. M. Lucas, "Evolving robust and specialized car racing skills," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 1187–1194.

[12] R. De Nardi, J. Togelius, O. E. Holland, and S. M. Lucas, "Evolution of neural networks for helicopter control: Why modularity matters," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 1799–1806.

[13] J. Togelius, "Evolution of a subsumption architecture neurocontroller," *Journal of Intelligent and Fuzzy Systems*, vol. 15, no. 1, pp. 15–20, 2004.

[14] T. Thompson and J. Levine, "Scaling-up behaviours in evotanks: Applying subsumption principles to artificial neural networks," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 159–166.

[15] J. Schrum and R. Miikkulainen, "Evolving multimodal behavior with modular neural networks in ms. pac-man," in *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2014, pp. 325–332.

[16] R. Calabretta, S. Nolfi, D. Parisi, and G. P. Wagner, "Duplication of modules facilitates the evolution of functional specialization," *Artificial life*, vol. 6, no. 1, pp. 69–84, 2000.

[17] J. Clune, J.-B. Mouret, and H. Lipson, "The evolutionary origins of modularity," *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 280, no. 1755, p. 20122863, 2013.

[18] J. Secretan, N. Beato, D. B. D Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley, "Picbreeder: evolving pictures collaboratively online," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 1759–1768.

[19] J. Clune and H. Lipson, "Evolving 3d objects with a generative encoding inspired by developmental biology," *ACM SIGEVOlution*, vol. 5, no. 4, pp. 2–12, 2011.

[20] M. Gallagher and M. Ledwich, "Evolving pac-man players: Can we learn from raw input?" in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. IEEE, 2007, pp. 282–287.

[21] M. Wittkamp, L. Barone, and P. Hingston, "Using neat for continuous adaptation and teamwork formation in pacman," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 234–242.

[22] L. Cardamone, D. Loiacono, and P. L. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 2622–2629.

[23] ——, "Evolving drivers for torcs using on-line neuroevolution," *Illinois Genetic Algorithms Lab., Univ. Illinois at Urbana-Champaign, Urbana, IL, Tech. Rep*, vol. 2009008, 2009.

[24] ——, "Learning drivers for torcs through imitation using supervised methods," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 148–155.

[25] J. Munoz, G. Gutierrez, and A. Sanchis, "Controller for torcs created by imitation," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 271–278.

[26] N. Van Hoorn, J. Togelius, and J. Schmidhuber, "Hierarchical controller learning in a first-person shooter," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 294–301.

[27] K. Chellapilla and D. B. Fogel, "Evolving an expert checkers playing program without using human expertise," *Evolutionary Computation, IEEE Transactions on*, vol. 5, no. 4, pp. 422–428, 2001.

[28] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.