To Plan or Not to Plan: Lessons Learned from Building Large Scale Social Simulations

Anton Bogdanovych ✉ and Tomas Trescak

MARCS Institute for Brain, Behaviour and Development,
School of Computing, Engineering and Mathematics,
Western Sydney University, NSW, Australia
a.bogdanovych@westernsydney.edu.au; t.trescak@westernsydney.edu.au

Abstract. Building large scale social simulations in virtual environments requires having a large number of virtual agents. Often we need to simulate hundreds or even thousands of individuals in order to have a realistic and believable simulation. One of the obvious desires of the developers of such simulations is to have a high degree of automation in regards to agent behaviour. The key techniques to provide this automation are: crowd simulation, planning and utility based approaches. Crowd simulation algorithms are appropriate for simulating simple pedestrian movement or for showing group activities, which do not require complex object use, but are not suitable for simulating complex everyday life, where agents need to eat, sleep, work, etc. Planning and utility based approaches remain the most suitable for this situation. In our research we are interested in developing advanced history and cultural heritage simulations and have tried to utilise planning and utility based methods (the most popular one of which is used in the game "The Sims"). Here we examine pros and cons of each of the two techniques and illustrate the key lessons that we have learned with a case study focused on developing a simulation of everyday life in ancient Mesopotamia 5000 B.C.

1 Introduction

Using Virtual Reality for reconstructing sites of high historical or cultural significance and showing how these sites were enacted in the past is becoming more and more popular. 3D modelling helps to visualise historical buildings and artefacts that at present only remain in the form of significantly damaged architectural structures or museum exhibits scattered around the world. Populating the reconstructed 3D environments with virtual agents helps to illustrate how these buildings and objects were used by people and to highlight the key technological or cultural aspects of the simulated society.

Supplying such Virtual Reality simulations with virtual agents that are capable of convincing and historically or culturally authentic behaviour beyond simple crowd simulation algorithms is difficult and costly. Virtual agents must be able to play different social roles, adhere to social norms, actively use surrounding objects, interact with other agents and even engage into interactions with humans. Modern video games are a good illustration in regards to the potential of having such simulations, but the cost of developing video games is very

J. Beskow et al. (Eds.): IVA 2017, LNAI 10498, pp. 53-62, 2017.
DOI 10.1007/978-3-319-67401-8_6

high. For example, the estimated cost of developing Crysis 3, one of the popular modern video games, is $66 Million [5]. Such a level of spending is not feasible when it comes to research simulations.

In this paper we discuss different techniques that could help to make agent-based historical, cultural and other kinds of virtual simulations (which we call "social simulations") more affordable by automating the process of populating a virtual environment with a large number of virtual agents. One of the key research questions that arises with building such social simulations is how to supply virtual agents with goals and plans for achieving these goals in a believable and culturally/historically authentic manner? In regards to supplying agents with complex goals, which can consistently lead to believable behaviour in a social simulation, one of the key approaches is to develop some sort of a computational simulation of agent needs (similar to Maslow's pyramid of needs [8]) and generate goals in response to a particular need requiring satisfaction. Each need can be seen as a reservoir that is replenished (satisfied) after performing the relevant action (e.g. sleeping to reduce fatigue) and depleted in response to other actions (e.g. walking will increase fatigue).

As for developing plans (sequences of actions that lead to satisfying these generated goals), there are two popular approaches that game developers and academics use: AI Planning [7] and utility based methods, the most popular one of which is used in the game "The Sims" [1]. In our work we have developed a computational simulation of needs and have applied both utility based methods and AI planning for making our agents satisfy those. The key objective of this paper is to share the lessons learned form applying these techniques and explain what are the pros and cons of both when applied to building social simulations.

The remainder of the paper is structured as follows. In order to facilitate a reader's understanding of the discussed techniques we first present an example scenario related to building a social simulation in Section 2. In Section 3 we explain how a utility based approach similar to the one used in The Sims can be used for social simulations. Section 4 explains the use of AI planning for need satisfaction. Section 5 outlines the results of our comparative study and analyses the pros and cons of each of the needs satisfaction methods. Finally, Section 6 provides a summary of lessons learned and concluding remarks.

2 Example Scenario: Ancient Mesopotamia 5000 B.C.

To illustrate the issues involved in building social simulations and discuss how it is possible to automate agent behaviour in those, we suggest to consider the simulation of everyday life in ancient Mesopotamia around 5000 B.C.[1] shown in Figure 1. For this simulation we have reconstructed 3 small ancient settlements using the settlement maps produced by the archeologists from [3].

Based on the knowledge obtained from history experts we could design the following simplified scenario portraying a day of an average citizen: an average

[1] Accompanying video is available at: https://youtu.be/5yEF2A7LEL0

Fig. 1. Social Simulation Example: Everyday Life in Ancient Mesopotamia 5000 B.C.

agent in our simulation should start its day at around 6AM in the morning. Soon after waking up the agent would eat breakfast by preparing the food from storage or obtaining food through work. The agent would eat 4 times a day with intervals between meals being close to 6 hours. In-between meals the agent would work to satisfy immediate hunger or comfort. There is hardly any recreation time. If there is, the agents should explore the city or chat with each other.

The type of work an agent must perform depends on the social role this agent plays in the simulation. For simplicity, consider a scenario where each of the agents plays one of the following 4 social roles: Fisher, Baker, Potter and Shepherd. Figure 2 outlines the key tasks that these agents must perform at work: milking sheep, making pots, baking bread and fishing.

The first step of building the simulation involved modelling the buildings and the settlement layout based on the results of archaeological excavations and information provided by subject matter experts. The next step was to manually design the appearance of the base population of 2 agents. Treating the appearance of the agents from the base population as genetic code allows to automatically generate a desired number n of the simulation inhabitants following the approach in [11]. Given that each of these agents must play one of the aforementioned 4 roles we can either equally allocate the generated agents into the given 4 roles ($n/4$ - shepherds, $n/4$ - potters, $n/4$ - bakers and $n/4$ - fishers) or come up with a way of specifying the role distribution in the simulated society (e.g. 20% shepherds, 20% potters, 30% bakers and 30% fishers).

The key question this paper tackles is: once an agent has been created and has been assigned with a particular role how can we make this agent automatically generate goals and plans, so that it simulates believable behaviour consistent with the aforementioned scenario?

Fig. 2. The Four Agent Roles: Shepherd, Potter, Baker, Fisher

3 Utility Based Method: Replicating the AI of The Sims

One of the most popular video games that simulates agent behaviour through modelling needs is "The Sims". Due to its commercial nature it was difficult for us to obtain a detailed explanation of the process agents in The Sims game use to address their needs, as most of such information is considered a trade secret by game developers and they are often not interested in sharing it with the public. In our exploration we had to rely on conference presentations, blogs, videos and white papers. The most comprehensive formal explanation is available in [1].

The essence of the approach taken by The Sims is that every agent is supplied with a set of needs. These needs do not decay uniformly, but their decay rate is determined by functions similar to those shown in the Figure 3. Each of the needs is represented by a value from the $[-100 \cdots + 100]$ interval, where +100 means that the need is 100% satisfied and -100 means that the need is 100% suppressed. The values for each need decrease over time as per Figure 3 and can be increased through interactions with objects that can satisfy them.

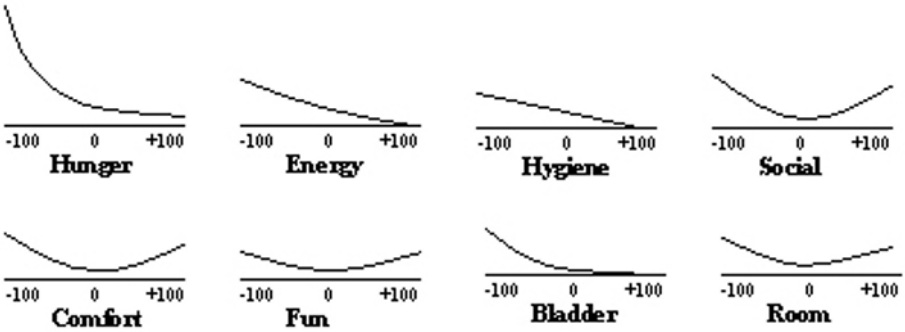

Fig. 3. Need Decay and Happiness Weights in The Sims [1]

There are no explicit goals that the agents have to generate and then pursue. Instead, the agents are driven by their desire to improve their mood. The mood of an agent is simply a sum of all the numeric values for agent needs as: $Mood = Hunger + Energy + Comfort + Social + Bladder + Room + Fun + Hygiene$.

Most of the "intelligence" in this game is stored in the objects rather than in the agents. Every time an agent finishes performing some action in response to an attempt to improve its mood it recomputes the current mood and then acts based on the information communicated by the surrounding objects. Some of the objects are capable of modifying the agent mood, so such objects contin-

uously broadcast the details of how they can address each of the agent needs. An example that is relevant to our scenario from Section 2 is shown in Figure 4.

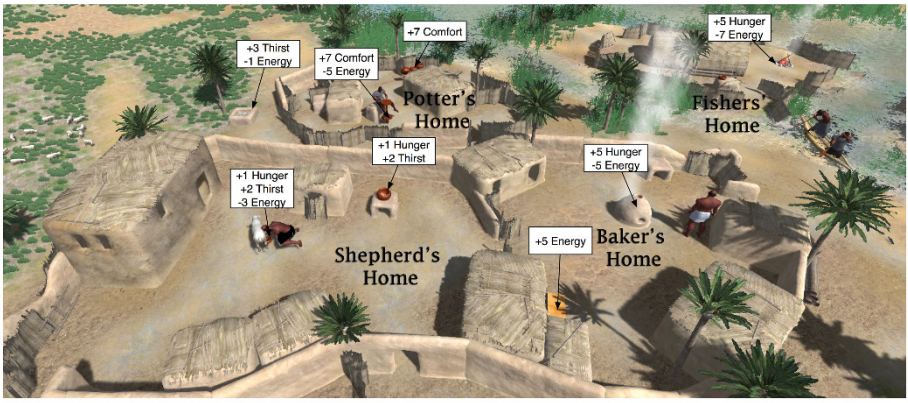

Fig. 4. The Overview of the Agent Architecture

To decide which action to perform next, an agent would compute the potential mood value using the information about the possible need changes broadcasted by every object and would then choose to interact with the object that provides the highest value of the resulting mood. Once the agent approaches an object and communicates that it wants to interact with it, the object would send a recipe (in our case a finite state machine) that prescribes what the agent must do to update its mood in accordance with the broadcasted values.

To avoid inter-agent conflicts in our implementation each agent owns a particular set of objects and only the owner can can interact with those. In our scenario from Figure 4, for example, all the objects located in the Shepherd's Home are owned by the shepherd agent and cannot be used by othes. There are also shared objects, which have no owner and can be used by any agent. An example of a shared object is the pot in the Potter's Home. This pot can be used by any agent and would increase the comfort of this agent by +7, but the supplied finite state machine would request from the agent using it to create another shared object of type food and leave it in exchange for the pot. Once an agent commits to interact with an object it can no longer stop the interaction and it is no longer possible by other agents to interact with this object.

Concerning the scenario from Figure 4, if an agent plays the role of a shepherd it can satisfy its hunger (+1) and thirst (+2) by interacting with a sheep. Once it approaches the closest sheep it must follow the actions expressed in the finite state machine that it supplies. This would involve approaching a pot, carrying the pot to the sheep, playing the milking animation, playing the drinking animation and then returning the pot to its original place. This would result in the decay of energy (-3). Alternatively, if there is milk available in storage this agent could choose to interact with the corresponding pot that would satisfy the agent's hunger (+1) and thirst (+2) with no energy decay. All agents can satisfy their thirst (+3) by using the shared object "well", but they will loose some energy (-3). A baker agent could satisfy its hunger (+5) by interacting with a stove. This

would involve collecting the wheat in the field, preparing the dough, putting it in the stove and then eating the bread. Fishers could satisfy hunger (+5) through a fire place. This object would make them collect their fishing gear, walk to the boat, board it, play the fishing animation, return back with the fish, cook it on the fire and then eat it. Potters can not directly satisfy their hunger, but could only satisfy it if they have stored food in their possession. They can obtain stored food through trade with agents of other 3 roles. These agents would require pots from the pot maker to address their comfort (+7) and would trade the food they produce for pots. The comfort need is addressed by all agents through having food in storage. The energy need is satisfied through sitting down or sleeping. The social need is addressed by talking with the closest agent.

4 AI Planning in Social Simulations

An alternative approach to the utility based needs satisfaction is to make agents generate a goal (e.g. make the value of hunger greater than +50) every time a given need reaches a critical value (e.g. hunger = 0) and then dynamically build a plan that can satisfy this goal using AI planning [7]. The key idea behind planning is to annotate every action an agent is capable of performing with pre-conditions (in our simulation this is the state the agent must have in order to perform this action) and post-conditions (how the state of the agent would change if a particular action is performed). Once the actions are annotated, an agent can dynamically obtain a sequence of actions that leads to satisfying the generated goal by performing a search through the action space.

The key traditional benefit of using planning in video games is that the resulting agent behaviour appears diverse and dynamic, because agents can easily integrate rapid changes in the environment (e.g someone taking away an object that is necessary for completing a task) and find an alternative if such alternative exists. Additionally, game developers are no longer required to manually design plans (e.g. finite state machines embedded in the game objects) to specify how particular objects are to be enacted. Instead, the agents are capable of dynamically building such plans. But those benefits come at a cost of reduced performance, as every time an agent must make a decision it has to conduct an exhaustive search for a plan instead of simply executing a finite state machine.

To illustrate the benefits of planning for social simulations Figure 5 shows a simplified example where an agent with the role "Potter" and whose current state includes "HasWater" constructs its plan from the list of the following actions:

- FindWater (**pre-condition** ="NoWater", **post-condition** ="HaveWater")
- Work (**pre-condition** ="HaveClay", **post-condition** ="HavePot")
- MakeClay (**pre-conditon** ="WaterInClay, **post-condition** ="HaveClay")
- AddWater (**pre-condition** ="HaveWater, **post-condition** ="WaterInClay")
- Trade (**pre-conditon** ="HavePot", **post-condition** ="HaveFood")

Through analysing pre-conditions and post-conditions of each of these actions the agent is able to search for a plan that leads to obtaining food through trade and involves creating a pot and then exchanging it for food. The resulting plan is: AddWater→MakeClay→Work→Trade→HaveFood. This search starts with

Fig. 5. PotMaker Planning Example: AddWater → MakeClay → Work → Trade.

the agent generating a goal "HaveFood". It can then find "Trade" as an action that has this goal state as a post-condition. This backwards search continues until the agents finds a sequence of actions that leads from its current state "HaveWater" to the goal state "HaveFood". If at some stage the water pot is removed from the agent's home its state is updated to "NoWater" and the precondtion "HaveWater" of the "AddWater" action will not be satisfied, so the plan for the "HaveFood" goal would need to start from the "FindWater" action.

Due to performance issues in the vast majority of the games where AI planning is used the actual number of agents that do planning is very small [2]. In social simulations we need hundreds of agents, so this is a problem, but, many agents in social simulations would have very similar goals and would frequently need to repeat similar routines, so there is potential for plan reuse.

There are different ways one can do planning, but the most popular approach used by the gaming community is the goal-oriented action planning (or GOAP) [9]. We have modified the classical GOAP implementation to fit our purposes (e.g. to include dynamic heuristics, allow for cycles in plans, work with real numbers instead of binary predicates, etc.), but even those optimisations did not help to reach real-time performance. Moreover, complex agent behaviour was difficult to structure using the classical form expected by GOAP. As a result, we have considered the use of Hierarchic Task Networks (HTN) [6]. In our prior work we have used Electronic Institutions [4], the structure of which adheres to HTN, as the way of expressing norms in social simulation, so we were able to combine GOAP and HTN via Electronic Institutions. While an average plan generation using classical STRIPS (for 22 actions) was around four minutes, with Electronic Institutions we brought it down to 100 milliseconds. Further details of integrating Electronic Institutions into our simulations are beyond the scope of this paper, but curious readers would find further details in [10]. What is important for this paper is that through integrating Electronic Institutions we were able to reach real time performance when planning with many agents.

5 Experiments

We have conducted a series of experiments using the scenario described in section 2 with the aim to understand the key visual and statistical differences between AI planning and The Sims approach. These experiments where performed on an Alienware 7 laptop computer with the screen resolution of 1027x768.

Low performance is one of the key reasons for the limited use of planning in video games. An acceptable performance measure widely used in games devel-

opment is frame-rate (how many frames per second (FPS) can be shown during game play). The higher is the frame-rate the more fluent is the game experience. It is often considered unacceptable to have the frame-rate lower than 30 FPS.

The frame-rate is dependent on many factors: a number of visible actors, a number of polygons that must be rendered at a given moment, etc. These parameters are very specific to each particular game. So, it is difficult to have a comparison that is beneficial to wider audiences if we were to only do experiments with our particular game. Therefore, we have conducted most of our experiments in a simple 2D environment where agents are represented as circles and objects they interact with represented as cubes. In this way we can eliminate game specific performance issues that are not related to planning.

Figure 6 shows a graph that represents the relationship between the frame-rate (vertical axis) and the number of agents (horizontal axis) in the Sims implementation. As can be seen from this graph, we were able to work with over 4000 agents in a 2D simulation and still have the comfortable frame-rate of 30 FPS. We have also measured the frame-rate in the 3D gaming environment outlined in Figure 1. There we had around 350 agents visible to the player at the frame-rate of 30 FPS. Having 400 agents reduced the frame-rate to 25 FPS.

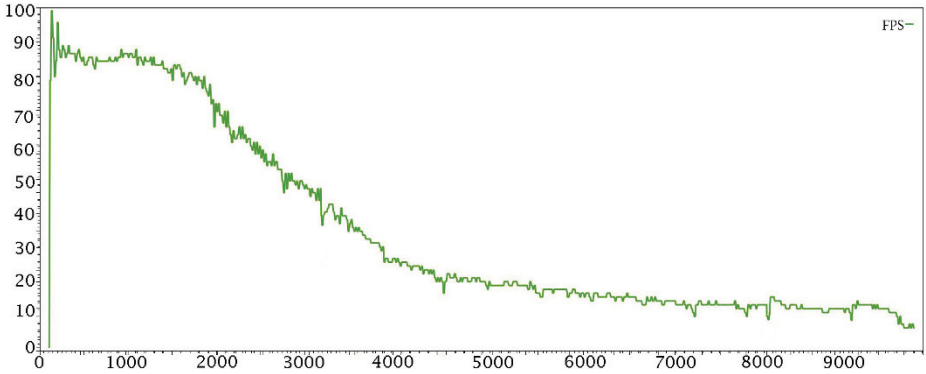

Fig. 6. Agents Controlled by a Sims like Model: Framerate vs Agent Number

Figure 7 shows the results we were able to achieve in the 2D simulation using AI Planning. Here we were only able to generate 105 agents and have the frame-rate about 30 FPS and then it would start to sharply decline. In the 3D simulation we could only have 50 agents that use planning at 30 FPS.

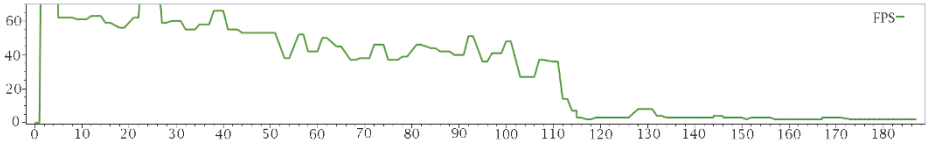

Fig. 7. Agents Controlled by a Classical Planner: Framerate vs Agent Number

These results were not very encouraging, so we have decided to modify the planning mechanism of our agents by supplying global plan memory (cache). This memory is used to routinely store the details about the current state of the

planning agent, its goal, role and the generated plan. Before generating a new plan we were then checking whether there is a record in memory that contains the plan for an agent with the same role, goal and state. If such record exists then we would execute the recorded plan instead of searching for a new plan.

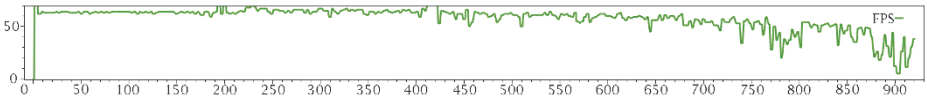

Fig. 8. Agents Controlled by a Planner with Memory: Framerate vs Agent Number

Figure 8 shows the results of using planning with memory within our 2D simulation. Unlike video games, social simulations are quite repetitive, so using plan memory resulted in significant performance improvements. We were able to achieve the performance of 30 FPS while having 770 agents present in the 2D simulation and close to 200 agents present in the 3D simulation at 30 FPS.

6 Discussion and Conclusion

We have compared two different approaches to automating agent behaviour in large scale social simulations. Both approaches relied on first building a computational model of needs and then having these needs controlling the actions the agent would choose to perform. The first approach to need satisfaction that we tested was the utility based method from The Sims game. The benefit of this method is that it is relatively simple to implement and the resulting performance is very high, potentially allowing to have thousands of agents in a simulation. But the key drawbacks of this approach are the following: the behaviour of the agents appears scripted, it is very difficult to come up with correct values for need changes that should occur when an agent interacts with an object that could satisfy them (e.g. Hunger +5 and Energy -7 offered by the fisher's fire place). If it is required to portray limited resources in the environment then The Sims implementation could be associated with believability issues, as lack of resources might lead to agents not being able to complete their plans and all resources that they use could be irreversibly lost as the fact of agents obtaining those cannot be reflected in their state. The Sims approach also has scalability problems. When adding new objects these numbers must be recomputed across the board, new finite state machines must be designed to support utilising this new object and some of the existing state machines might have to be modified if the use of this object is required in the corresponding plans. Another significant issue was supporting agent interactions with one another. The Sims approach is useful when agents act in isolation, but if they have to conduct complex tasks together it is difficult to have them synchronised without breaking believability.

The use of planning allowed to achieve more believable and emergent behaviour and solve the aforementioned synchronisation issues. It was also much easier to design the system and modify it by adding new objects. But the use of planning was associated with a significant performance loss. Through various kinds of optimisation (integrating Electronic Institutions and plan memory) we

were able to achieve having 770 agents in the 2D simulation running at 30 FPS and close to 200 agents running in our particular 3D simulation at 30 FPS. There is a lot of room to optimise our 3D simulation and with improved hardware we could, of course, generate more agents. We could also overcome the agent number limitation by using classical gaming strategies like hiding agents when they are not in the line of sight or by only performing planning for the agents that the user is currently looking at and use a simpler technique.

Our concluding suggestion would be to certainly focus on planning and work on strategies to optimise it if your goal is to build a research simulation that is similar to the scenario described in this paper, where low frame rates are not very critical and where you can potentially use a powerful super computer for running the simulation. But if you are building a commercial game that needs hundreds or thousands of agents running on a personal computer then performance problems associated with planning would outweigh the believability benefits and it is better for you to follow The Sims approach to needs satisfaction. Finally, for simulations with a high degree of agent interaction (e.g. trading, working together) planning is a better choice due to increased believability and reduced design complexity.

References

1. AI Game Programmers Guild, http://gameai.com/wiki/index.php?title=The_Sims: The Sims (2011)
2. Champandard, A.J.: In-Depth Study of Planning in Top AAA Games. AiGameDev.com (February 28) (2014)
3. Crüsemann, N., van Ess, M., Hilgert, M., Salje, B.: Uruk. 5000 Jahre Megacity (2013)
4. Esteva, M.: Electronic Institutions: From Specification to Development. Ph.D. thesis, Institut d'Investigació en Intelligència Artificial (IIIA), Spain (2003)
5. Gauder, J.: Crysis 3 cost $66 million to make, can next gen sustain such budgets? GameChup Video Games News at http://www.gamechup.com/crysis-3-cost-66-million-to-make-can-next-gen-sustain-such-budgets/ (2013)
6. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann Series in Artificial Intelligence, Elsevier (2004), https://books.google.com.au/books?id=eCj3cKC_3ikC
7. Hendler, J.A., Tate, A., Drummond, M.: AI planning: Systems and techniques. AI magazine 11(2), 61 (1990)
8. Maslow, A.H.: A theory of human motivation. Psychological review 50(4), 370–396 (1943)
9. Orkin, J.: Applying goal oriented action planning in games. In: AI Game Programming Wisdom 2, pp. 217–229. Charles River Media (2002), http://web.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf
10. Trescak, T., Bogdanovych, A., Simoff, S.: Populating virtual cities with diverse physiology driven crowds of intelligent agents. In: Proceedings of the Social Simulation Conference (SSC 2014). pp. 275–286 (2014)
11. Trescak, T., Bogdanovych, A., Simoff, S., Rodriguez, I.: Generating diverse ethnic groups with genetic algorithms. In: Proceedings of the 18th ACM symposium on Virtual reality software and technology. pp. 1–8. VRST '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2407336.2407338