Integrating a simulation model as an architectural component of a game

André Koscianski[1], Guilherme T. S. Abreu[2], Luiz G. M. Padilha[2]

[1]PPGCC-PG – Universidade Tecnológica Federal do Paraná (UTFPR)
R. Dr Washington S. Chueire, 330 Ponta Grossa, PR - Brazil

[2]Department of Computer Science – UTFPR.

koscianski@utfpr.edu.br, {guiabr,luipad}@alunos.utfpr.edu.br

Abstract. *Many computer games involve the simulation of real-world phenomena, mechanisms, and behavior of beings. Examples vary from simple Physics found in the platform style to characters that learn and adapt according to user choices. Another field that uses computer models is criminology, a theme that is also a starting point for many games. This paper studies the integration of a (serious) criminal simulation model and a computer game, keeping the two perspectives separated; the model functions as an add-on that reshapes parts of the game.*

Keywords— *simulation model, computer game, software architecture.*

1. Introduction

Simulations found in games vary from simple physics of animated cartoons (Super Mario) to detailed description of aerodynamics (Flight Simulator). Another variation is the representation of historical facts (Les Campagnes de Napoléon); one of the challenges in designing this kind of title is to preserve historical accuracy but still offer a legitimate game experience [Dillon 2008].

The present work was born from two initially separated projects: a serious simulation model in the field of criminology, and a computer game with a hero (the player) who fights bandits in a city. The game was designed independently from the simulator, but should allow the integration of the criminal model as an optional element. The central problem in this context was to coordinate the architecture of both elements, game and simulator, accounting for mutual functional dependencies. The game does not depend on accurate descriptions of criminal activities or police routines. In the same line, an implementation of criminal theories is completely unconcerned with things as scores, sounds, graphical animations, user interaction and real-time execution. However, there are features that overlap since game and simulation, NPCs and agents, environment, and data maps, are elements that share a common ground. Both call for message

passing, state management, interaction, and spatial and temporal coordinates. In the present case, there was a focus on the game engine, and the situation would be different if the project started with agent-centered platforms such as Netlogo, Repast, or Jacamo.

2. Background

The development process of games has important differences compared with traditional software [Aleem et al 2019]. Architectural descriptions present an inherently partitioned view of software and the relations between parts.

Component-based development is a technique well adapted to the needs of the game industry; it supports the variety of functional requirements present in this type of software, provides flexibility to create new elements, and supports reuse that contributes to lower costs [Freitas et al 2012, Van der Vegt et al 2016]. The separation between game and simulation model resembles the Model-View-Controller architecture, where the simulator would correspond to the controller. However, both the crime model and the game code are responsible for different aspects of the functioning of the agents. Some techniques that come to mind in this scenario are aspect-oriented, component-based, and incremental design; nonetheless, the crosscutting character of requirements may not be addressed correctly under classical views [El-Hokayem et al 2018]. One way to frame the problem is by means of feature-oriented software development (FOSD), a paradigm that highlights feature modeling, dependency, and interaction and fits some of the gaps of the other cited techniques [Apel and Kästner 2009]. In short, features are viewed as determinant of structure and not a by-product of functions and behaviors. In that view, features drive both design and implementation of software. Figure 1 illustrates the concept applied in the present case.

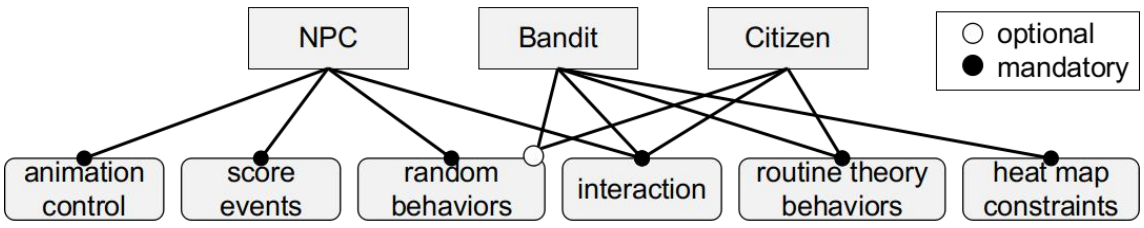

Figure 1. A partial feature model of the project (using [Apel and Kästner 2009] notation).

Figure 1 shows that, while a bandit maps to an NPC in the game, some elements of that entity also belong exclusively in the simulation. One way to implement separated requirements in a completely transparent way might be dependency injection with the support of reflection [Passos et al 2010]; in the present case, scripts have minimal awareness about each other and pilot variables and states to accomplish their objectives.

2.1. Crime Simulation Models

Crime simulators can represent the main features of the behavior of individuals to predict unlawful actions [Malleson et al 2013]. Some elements found in such models are the representation of people's routines, criteria and processes used to choose a course of action, and reinforcement mechanisms [Birks and Elffers 2014]. Agent-Based Modeling (ABM) is an established approach in criminology [Groff et al 2019], and it can also be used to model a game architecture in terms of its characters [Rhalibi and Merabti 2005].

In this project, bandits in the original game concept might follow random rules concerning entering the scene and moving around; the simulation would refine this information to drive agents according to a logic prescribed by criminal models [Devia and Weber 2013].

3. Development

The implementation of a proof-of-concept was carried out with the Godot game engine, with a simulated city in isometric projection. Besides the city map, the simulator and the game must share a common view of the state of the world. There are two general options: either the two components run separate models and synchronize information using some protocol, or they function on a client-server basis. The crime model requires agents to navigate through a city; many aspects of this function, like routing, kinematics, and collision detection, are naturally supported by game engines like Godot. At the same time, the very presence of those agents is a requirement of the game itself. Because of this, the game code was chosen as a first layer to breath life into the characters. The simulator delegates all those responsibilities to the game code, which, in turn, may rely on the simulator to obtain events signaling unlawful actions.

The NPCs were implemented with state machines. The decision to pursue a citizen depends on the rules implemented in the simulator. The model from Devia and Weber (2013) uses a probability distribution and the mood of a bandit to define the likelyhood of an assault. This model was the basis for the simulation component.

Periodically, the simulator is called by the game engine and runs an analysis of the state of the scenery. If the right conditions are met, the state of a criminal agent is switched to 'Chase'. As shown in Figure 2, the communication between the components begins with a message from the game, notifying the simulator about the current time. This message is generated in the game loop. The simulator updates its internal clock and periodically queries the game. It sends the position of a bandit and receives in return a list of civilians withing a given radius, with the closest one in the first position. For this implementation, a list with a single element is the condition to switch the behavior of the bandit to the state 'Chase'. The movement of the NPCs is implemented in the game side; the interception of the victim

uses a traditional solution of following the target. Once this happens the bandit agent sends a message informing the simulator.

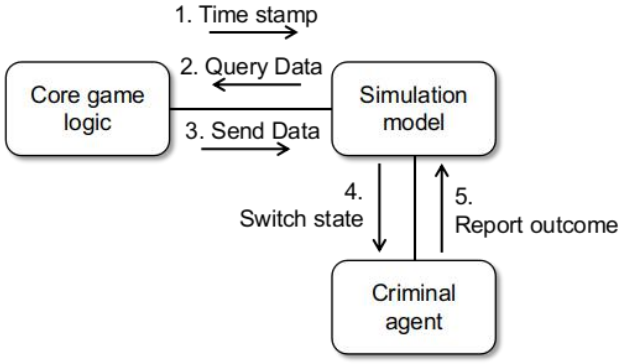

Figure 2. Communication diagram.

The initial version of the software implements simple random movement for regular citizens. The simulator chooses a destination and sends the information to the game code, which computes a path and moves the agent. A more detailed simulation model would define the trajectories according to predefined schedules and profiles like ‘student’, ‘worker’ and ‘common’, and routines such as ‘going to school’ and ‘return home’. The simulator can command these behaviors by sending messages with state changes and destinations to the NPCs in the game. To further improve authenticity, the agents may make random detours and pauses along their routes.

In our study using Godot, the simulator was coded as a regular object that, once instantiated, is activated (called) by the engine at each frame update. It has all the engine’s capabilities at its disposal, like the graphical design of scenarios and functions like path-finding. Since all objects (NPCs) are public, the simulator script can directly manipulate states and variables, overruling the regular (not-model-aware) game code. Since this interference must not be treated as an error, the NPC code must leave that possibility open; for instance, state machines should not forbid foreign (unknown) states, which a given simulator implementation can use to bypass the game logic.

3.1. Game Mechanics

An essential aspect of the game was the definition of a storyline that could accomodate the autonomous activities of agents. A simple concept was created; the basic mechanics are represented in Figure 3 using a Machinations diagram [Dormans 2012].

The diagram in Figure 3 starts with the generation of a crime event by the simulator. This event triggers the action of a criminal agent, and this generates a warning to the player. The possibility of a bandit not accomplishing the action was left for future versions of the software: it might be a consequence of the presence of police agents patrolling the city. Following the diagram, the occurrence of a robbery

decreases the player's score; on the contrary, the score increases when the player catches a criminal. After a certain number of victories (ten in this design), the player is granted access to a new level. The game leads the player to enter a procedure-generated maze where several bandits and a final boss are hidden.

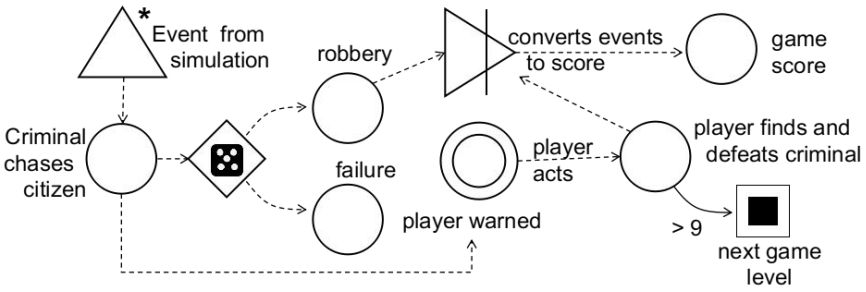

Figure 3. **Game Mechanics.**

4. Conclusions

Simulation is intrinsic to most games, with Physics being the main example and requirements varying from crude kinematics to sophisticated dynamics. Normally a simulation is conceived as part of the game logic. A different perspective is keeping a simulation model separate from the game concept. In this project, the simulator functions as a graft that can overcome the game control logic. At the same time, the simulator also plays the role of a client that makes use of NPC functions, like movement; an entity as 'Citizen' has features from two perspectives: game/NPC and simulation/agent. Techniques such as design by contract and data abstraction make it possible to reduce coupling up to the point where a pluggable architecture is defined. In the present case, manipulating object properties was the key to addressing the issues illustrated in Figure 1. On other implementation platforms, mechanisms such as reflection might be explored to expose interfaces that would be queried and activated from distinct software modules.

Several functions typically found in game help implement agent-based simulations; this includes movement, collision detection, and handling groups of characters. Visualization is another vital function, allowing modelers to 'feel' and validate the adequacy of behaviors. Game engines support all of those features using highly efficient algorithms, and solutions like the one discussed here may contribute to decouple dependencies from scientific applications.

References

. A. Dillon, "Signifying the west: colonialist design in Age of Empires III: The WarChiefs," Eludamos: Journal for Computer Game Culture, vol. 2, no. 1, pp. 129–144, 2008.

S. Aleem, L. F. Capretz, and F. Ahmed, “Critical success factors to improve the game development process from a developer’s perspective,” Journal of Computer Science and Technology, vol. 31, no. 5, pp. 925–950, 2016.

L. G. Freitas et al., “Gear2d: an extensible component-based game engine,” in Proceedings of the International Conference on the Foundations of Digital Games, 2012, pp. 81–88.

W. Van der Vegt, W. Westera, E. Nyamsuren, A. Georgiev, and I. M. Ortiz, “RAGE architecture for reusable serious gaming technology components,” International Journal of Computer Games Technology, vol. 2016, 2016.

El-Hokayem, A., Falcone, Y., & Jaber, M. (2018). Modularizing behavioral and architectural crosscutting concerns in formal component-based systems–Application to the Behavior Interaction Priority framework. Journal of logical and algebraic methods in programming, 99, pp. 143-177.

Apel, S., & Kästner, C. (2009). An overview of feature-oriented software development. J. Object Technol., 8(5), 49-84.

Passos, E. B., Sousa, J. W. S., Clua, E. W. G., Montenegro, A., & Murta, L. (2010). Smart composition of game objects using dependency injection. Computers in Entertainment (CIE), 7(4), 1-15.

N. Malleson, A. Heppenstall, L. See, and A. Evans, “Using an agent-based crime simulation to predict the effects of urban regeneration on individual household burglary risk,” Environment and Planning B: Planning and Design, vol. 40, no. 3, pp. 405–426, 2013.

D. Birks and H. Elffers, “Agent-based assessments of criminological theory,” Encyclopedia of Criminology and Criminal Justice, pp. 19–32, 2014.

E. R. Groff, S. D. Johnson, and A. Thornton, “State of the art in agent-based modeling of urban crime: An overview,” Journal of Quantitative Criminology, vol. 35, no. 1, pp. 155–193, 2019.

A. El Rhalibi and M. Merabti, “Agents-based modeling for a peer-to-peer MMOG architecture,” Computers in Entertainment (CIE), vol. 3, no. 2, pp. 3–3, 2005.

N. Devia and R. Weber, “Generating crime data using agent-based simulation,” Computers, Environment and Urban Systems, vol. 42, pp. 26–41, 2013.

J. Dormans, Engineering emergence: applied theory for game design. Amsterdam: Universiteit van Amsterdam, 2012. [Online]. Available: https://hdl.handle.net/11245/1.358623