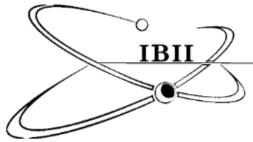

Journal of Education and Social Development, 2019, 3–2
December. 2019, pages 35-40
doi: 10.5281/zenodo.3591893
http://www.ibii-us.org/Journals/JESD/
ISBN 2572-9829 (Online), 2572-9810 (Print)

Case Study: Designing and Developing a Game Prototype

Vi Ta, Ling Xu *

Department of Computer Science and Engineering Technology, University of Houston – Downtown, USA.

*Email: xul@uhd.edu

Received on 12/19/2019; revised on 12/23/2019; published on 12/23/2019

Abstract

Gaming industry and game development are quickly growing. However, learning game development is often a challenge for Computer Science (CS) students. In this paper, we introduce a student project for designing and developing a game prototype with details. In the project we combine a few essential game features by implementing multiple play modes, NPCs, and randomness and varieties. We especially introduce the use of artificial intelligence (AI) in simulating the behaviors of intelligent agents, which we believe can be beneficial to the understanding of AI applications in gaming. In addition, the design ideas and implementation details can be helpful to the design of student game projects in CS teaching and may also inspire the development of other games.

Keywords: Game Development, Tower Defense Game, Artificial Intelligence

1 Introduction

Video games own billions of players around the world. According to a recent report from Entertainment Software Association (ESA, 2019), about 65% American adults play video games. Gaming industry and game development have become a quickly growing business nowadays. Due to the popularity of game industries and tremendous job opportunities, college students show high enthusiasm in game playing and game development. As reported by ESA, 52% of gamers are college educated. The fact reflects the great needs from college students to learn gaming knowledge and corresponding developing techniques. On the other hand, because developing a video game usually requires multidiscipline knowledge such as computer programming, math, and arts, making game development a difficult task for many students. According to the author's teaching experience on university game programming courses, learning and independently designing and developing a single-player casual game within a four-month semester is still a challenge for many CS students.

Here we present a case study of a game development project at University of Houston - Downtown. In this project, a computer science student independently developed a game that involves the essential game components. The process of completing the project is also the process of learning the knowledge of game development. We tried to integrate necessary modules into one project, in order to make the student understand the abstract game concepts and programming methodology. We found that a carefully designed game project can motivate student learning interest and promote their critical thinking skills. At the same time, the game development can also facilitate the study of other fields, especially Artificial Intelligence (AI). AI emerged as a field in the 1950s. It is focused on mimic the cognitive functions of the human mind (Russell, 2009). Due to the width and the depth of AI theories, teaching AI algorithms usually is considered "painful" (Woolf et al., 2013). Since AI plays an important role in the game development, we also expect that the application of AI ideas in game design can make AI concepts concrete and vivid to students, thus strengthening their understanding and integration of the knowledge system.

2 Related Work

The task of the project is to develop a 2D tower defense game. Tower defense is a subgenre of the strategy games, where the player's goal is to defend her territories (or objects) from enemies (Avery, 2011). The player can achieve this goal by obstructing enemies' path with defensive objects, which will act automatically to attack or destroy enemies or hinder enemies. A game of a new take on the tower defense genre is the game Plants vs. Zombies by PopCap Games. In this game, the player uses plants (seed packets) which cost sun (the currency) to defend their house from hordes of zombies. Screenshots of the game are shown in Figure 1.

Our project attempts to give a refreshing take on tower defense with attractive features that captivated players within other popular games. One of the game features, the conveyor-belt levels in Plants vs. Zombies

Copyright © 2019 by authors and IBII. This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

is a feature incorporated into this game as a gameplay mode. In the conveyor-belt level, there is no sun cost (sun is used to use the seed packets), so plants that produce no sun is given. The player is given random seed packets via a conveyor-belt, this mechanism is called “Special Delivery”. Once the conveyor-belt is full, player cannot receive more seed packets, so the player must use up the remaining plants before new plants are delivered to the player. This game gained recognition and popularity for keeping the player engaged in addictive, attractive and unique gameplay. Because of the multitude of features presented in the game and the simplicity of the player goal in the game, a 2D tower defense game became the most feasible solution to create an engaging game within a given time frame as a student project.

Fig.1. The game Plants vs. Zombies: Normal level (upper); Conveyor-Belt level (lower).

3 Game Design

When designing the project, we intend to include the essential game elements proposed by Adams (Adams, 2014): play activity, pretended reality, goal, and rules. Play activity defines the level of participation from the player. Pretended reality is the environment created by the game developer, and the virtual world that the player will perceive once playing the game. Goal involves challenges to the game players in order to motivate the game playing. Rules provide instructions and restrictions to guide the playing activities. Our design of each element is described in the following.

- Goal

The virtual game world that we built includes characters and resource objects. The goal of the game is to defeat enemies and prevent them from crossing over the barrier. This requires the player to actively strategize the uses of resource objects, such as adding wall units to prevent the enemy moving. The player will combat three waves (invasions) of enemies per level and manage their resources to advance in the game until the game end conditions are met.

- Rules

In this game, the player can place resource objects (such as wall units and bombs) in the game world. Once player places a unit onto the game field, player cannot move it, until a wave of enemies is eliminated, and the next enemy wave starts. The player has three lives in each level, and the duration of a life (i.e. health points) will decrease if attacked by the enemies and increase with a healing resource object. The result of the game varies by the gameplay modes, i.e., Normal mode, Survival mode, and Surprise mode. Detail of the gameplay modes is introduced in 4.2.

- Play

The game included various gameplay modes, which provide challenges and variations for the player. The player would need to develop a strategy to place their limited resource objects, in order to defeat the enemies and protect their domain from the invasions. For certain levels in specific gameplay modes, the player would receive rewards for their progress.

- Pretended reality

Video games often contain various agents in the virtual game world. One typical category of agents is the non-player characters (NPCs). NPCs are autonomously computer controlled. Their behaviors in the game world are programmed by the game developer. As mentioned by Eichenbaum et al., the fun in playing video games is greatly affected by the procedure that players learn to succeed on “a set of tasks that are initially quite difficult” (Eichenbaum et al., 2014). Having rational NPCs that can respond based on the player’s actions is crucial to a game, because a player would become more involved in the game if there is a level of challenge and difficultly. If NPCs ignore player actions and do not act rationally (e.g. what a human may consider logical), the game will not be engaging or convincing, thus decreasing the player experience of fun playing. Towards that goal, we designed two types of intelligent agents (IAs), *Enemy* and *Ally*, that either act independently from player or dependently based on the player decisions and environment. The Enemy agents would attempt to hinder the player, whereas the goal of the Ally agents is to assist the player by seeking and collecting items for the player in order to defeat the Enemy agents. The enemy and ally IAs can move intelligently based on the programmed AI. The enemy wanders and avoids obstacles set by the player, while the ally moves towards the direction of its target.

Fig. 2. An Enemy agent (left) and an Ally agent (right).

4 Implementation

To begin the development of the game, the in-game objects must be defined. We use Processing, an open source computer language and integrated development environment (Reas and Fry, 2014). The programming can be implemented using the Java language.

4.1 Entities in the game

Within the game there would exist enemy units and player units. The player units consist of four types of units: ally, wall, trap, and golem. Constructing the player units has virtual costs in the game. We define four classes in Java for the four types accordingly.

- Ally units can seek items and attack enemies. The moving of ally units is not controlled by the player but the programmed game AI. They will support the player and act independent of the player actions.
- Wall units can stop enemies from moving momentarily and have high health points (HP). The wall unit is an inexpensive unit that allows the player to build their defenses.
- Golem units can attack enemies within a distance by causing harms (i.e. making enemies lose HPs).
- Trap units can instantly destroy enemies that step onto them. When normal enemy units encounter the pitfalls, they will be destroyed and removed from the game field. Player cannot remove a pitfall until the pitfall becomes ‘full’ (e.g. the maximum of enemy units in pitfall reached). The trap unit is uncommon to tower defense games, because enemies in traditional tower defense games do not cross paths with player units, trap units would not exist in that game environment. The trap unit is like the land-mine in *Plants vs. Zombies* which will detonate whenever enemies step on the trap. However, unlike the land-mine, the trap unit in the project can be used more than once.

In addition to the above four classes, we define the class for enemies. Enemies can attack and destroy player units. The enemy units have two types, normal and boss, which determine whether the enemy will be affected by the player units. Unlike normal enemies, the boss enemies cannot be defeated by a trap unit. The conclusion of the above entities/classes is shown in Table 1. A screenshot of the game scene is shown in Figure 3, where the bottom of the lower image shows the enemy units and the player units.

Table 1. Description of Entities

Entity	Description
Enemy	Ability: can attack player Attributes: normal enemy can be defeated by trap; boss enemy is immune to trap.
Ally Unit	Ability: can seek and attack enemies Attributes: has stamina, animation, and automatously moves.
Wall Unit	Ability: can stop enemies Attributes: has high health points (HPs).
Trap Unit	Ability: can instantly destroy enemies Attributes: has a counter for caught enemies and maximum capacity.
Golem Unit	Ability: can attack enemies within a distance Attributes: has a distance of influence

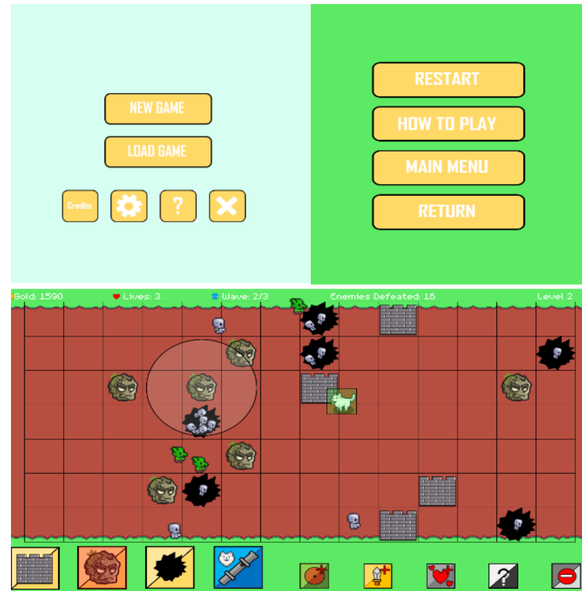

Fig. 3. Upper: game menu(s); Lower: a screenshot of the game scene.

4.2 Play Modes

In many games, there exists gameplay modes that give players a variety of choices to enhance the player’s gaming experience. In this game, we provide three gameplay modes: normal, survival, and surprise.

- Normal mode: this mode consists of the traditional gameplay of tower defense games. The player can choose from four types of units: ally, wall, trap, and golem. There can only exist one ally unit at any time in the normal mode. Placing a player unit onto the field has costs.
- Survival mode: In this mode, the resources are limited. The play has to manage the resources to avoid being defeated by enemies within a given time. After the initial resource items are used, no more items will appear. Like normal mode, player can choose their units, and there can only exist one ally unit at any given time. The more waves of enemies the player defeats, the greater the rewards. The survival mode challenges the player’s reaction and manipulating speed.
- Surprise mode: the player can only use the randomly generated units, such as initial resources and enemies. On one side, because it is possible to have multiple ally units, the player can be more powerful than in the other two modes. One the other side, the number of enemies and their types can make the game extremely challenging to the player. This mode provides most randomness and variations, giving surprises to the player.

In the normal and surprise modes, the player will win the game by making it to the end of the level with lives remaining. In the survival mode, the player tries to maintain a life for a period of time then win the game; it challenges the player to quickly manage their given resources and conquer as many enemies as possible.

4.3 Game AI

As pointed by Miikkulainen, “games are challenging yet easy to formalize, they can be used as platforms for the development of new AI methods and for measuring how well they work” (Miikkulainen, 2006). The game AI is an important aspect that we intend to explore in this project.

The Game AI in this project is focused on the behavior of the IAs. An object can change its state with time or invoked by an event. To implement the transitions from one state to another, the game employs a push-down automaton (PDA). A PDA is an extension of finite-state machine (FSM), however, a PDA can recall the history of states whereas an FSM cannot. In a PDA, the new state is pushed onto the stack, while keeping the previous state in memory. Here our intention to use PDA is that once the enemy and ally units have completed a state, we want them to revert to their previous state, which can be easily found from PDA.

The enemy units have three possible states: wander, attack, and flee. The start of the stack for enemies is the wander state. Once an enemy is in-range of a player unit the attack state will be pushed onto the stack, and then the enemy will attack. Once the attack state is complete, the state will be popped from PDA, and the enemy will resume the initial wander state that is on top of the stack. If the enemy has low HP, the flee state will be pushed, and the enemy will avoid player units if possible; the flee state will be popped if the enemy regains HP.

In the same way, the ally unit(s) have five possible states: idle, seek, wander, rest, and flee. The initial start stack for the ally unit is the idle state. For the player ally unit, the wander state is pushed whenever there are enemies out-of-range and on the field; meanwhile, the attack state is pushed whenever the enemy units are in-range of the ally unit. If items exist on the field and no enemies are in-range, the seek state will be pushed, and the ally unit will seek and collect items for the player. The ally unit will lose HP when it is not in idle state or rest state; when in rest state, the ally unit will gain HP. If the ally unit HP is less than the full HP bar, then the rest state is pushed onto the stack. The usual behavior pattern would be for the ally unit would for the ally unit to remain idle before wave starts and then seek objects and if enemies are in range, go attack enemies, and rest when out of stamina. The example stack is displayed in Figure 4.

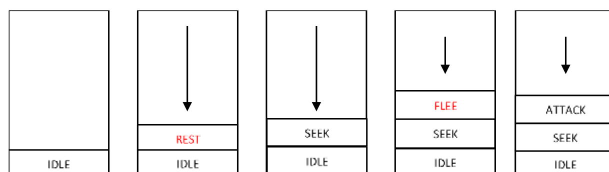

Fig. 4. Example Pushdown Automata for Ally Unit.

Based on the entities and the states, we define their movements. Here the Enemy units are the primary mass of objects (except in the surprise mode we may have multiple Ally units), and they must move reasonably by following some rules. Each enemy must behave uniformly when in a group, meaning not colliding with each other and moving in same direction. This intended behavior is called flocking behavior (Nara, 2012). We simulate the flocking behavior using the rules of boids (Reynolds, 1987) as the following (also see Figure 5).

- separation: the moving unit will steer to avoid crowding other units
- alignment: the moving unit will steer towards the average heading of the community of all units
- cohesion: the moving unit will steer to move towards the center of the community of all units

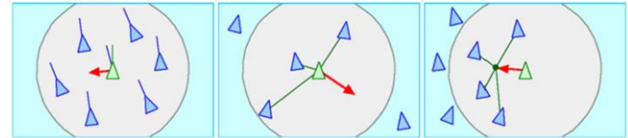

Fig. 5. Flocking behaviors (Reynolds, 1987): Alignment (Left). Separation (Center). Cohesion (Right).

To avoid having objects collide with each other, collision detection is implemented. Collision detection is used by NPCs (i.e. Enemy and Ally agents). Each NPC object will search an ArrayList (a dynamic array in Java) of objects and compare distances of the other objects and itself to determine if the object is too close to another object. First, the distance between two objects is calculated and if the distance is less than a specified value, then an opposing vector between the objects is found and added to the current object’s velocity.

The desired vector, or the opposing vector, is obtained by subtracting the current object’s position from the position of the detected object. The steer vector is also obtained, by subtracting the velocity vector of the current enemy object and the desired vector. After obtaining the vectors, the program sets the magnitude of the desired vector, then adds the steer vector to acceleration. Finally, the program updates the motion properties of the object calling the detection function.

With collision detection in the game, the player will perceive the NPCs as rational, since enemies would move away from player units and not walk over player units. If enemies collide with other objects, the enemy NPCs would not be considered intelligent or rational. After all, the rational action would be to move out of the way if an object is obstructing the way; which is why collision detection is necessary in game AI. Once the collision detection and avoidance are in place, enemy movement had to be fine-tuned within the array of enemy objects.

4.4 Player Choices and Game Balance

In addition to the game AI, we also implemented two components of the playful model proposed by Ferrara (Ferrara, 2011): player choices and game balance. Other components of the playful model include motivation, usability, and aesthetics. Here our rationale is that students have more space to explore the game design and AI programming in player choices and game balance, while other components focus more on the interface construction and artistic design.

In order to increase the weight of player choices, various limitations are applied to the player units. Adding limitations ensure that the player will not use only one unit over other units or develop a dominant strategy. A dominant strategy is that the one that can generate an overwhelming payoff compared with whatever other strategies the player may use (Reas and Fry, 2014). If the player can develop a dominant strategy to reach the goal, then the game becomes repetitive and eventually players

lose interest in the game. Different player units with different costs, abilities, limitations, can encourage the player to make meaningful choices and develop strategies to reach the game goal.

We apply limitations to player units, in order to achieve the game balance. Those limitations are based on four attributes: costs, cooldown time, attack ability, and attack power. For example, the player cannot use only the wall unit to win. Because the wall unit has no attack ability, the player can only defend for a short amount of time before the enemies destroy the walls and break through the player's defenses. Likewise, the player cannot use only the golem unit, since the golem has low attack power and requires the player to upgrade the attack power; the golems will not be able to destroy all enemies before the player runs out of resources to build more golem units. The ally unit is a powerful unit to support the player. However, it has a cooldown time, so the player cannot solely rely on ally to defeat all enemies. Similarly we set the cost limitation. The trap unit has the highest cost to use and a capacity threshold (i.e., the maximum number of enemies that the trap can destroy) before the player needs to upgrade the trap. If the player tends to rely on trap only, she will run out of resources at the beginning of the level and will not be able to regain resources if the traps are full, thus will slowly lose if too many enemies appear at once.

Although there are limitations to player units, there are opportunities for the player to offset the limitations in the core mechanics of the game. Through the upgrade system, the player can increase the player units' abilities and make them stronger than they were in the beginning. Each player unit has a set of possible upgrades (shown in Figure 6) that will increase the abilities of the unit, as follows:

- HP+: Increase unit's HP.
- Attack+: Increase unit's attack power.
- Range+: Increase unit's attack range.
- Evolve: Increase all abilities.
- Capacity: Increase maximum allowed victim count for the trap.
- Mystery: Random advantage or disadvantage to player.
- Remove: Remove the player unit from the field.

Fig. 6. Icons of Upgrades. From left to right: HP+, Attack+, Range+, Evolve, Capacity, Mystery, and Remove.

The upgrade list varies for each type of player unit. Wall and trap units do not have the ability to attack, so the 'Attack+' or the 'Range+' upgrade would not be available to these units. The 'Capacity' upgrade would only apply to trap units. The 'Remove' upgrade does not improve a unit's abilities; it removes the unit. Although 'Remove' is not actually an upgrade, since it makes the removal of a player unit easier, we also include it in the upgrade list. After the player upgrade units' skills, the cost (to upgrade) and the skill level will increase linearly and incrementally respectively.

On the other hand, in order to decrease predictability of the game for the player, we incorporate random elements in the game. Random elements are things out of the player's control, and their existence in the game will increase the variety of possibilities and fun of gameplay. First random element, the number of items on the field, is generated at the beginning of the level. Another random element is the selection of player units, available to the player in the surprise mode. Random elements also affect enemy movements at runtime, such as to decide an enemy's moving direction, or the number of enemies in each wave. Player can play the game repeatedly and get different results in each playthrough.

5 Conclusion and Future Work

In the above sections, we introduced designing and developing of a game prototype. In this project, we explored several features of game development, such as multiple game play modes, NPCs, randomness and varieties, and game AI. By thinking in terms of how the player would play the game or how to improve the player's experience in the game, the project fleshed out ideas that attempted to keep the player engaged in gameplay. When thinking as a developer, the goal is how to implement the functions and ideas in the core mechanics to achieve the desired modules. When there was a decent amount of functions in the game, the focus shifted to balance the game and work in the AI system with the other game features to create entertaining yet challenging gameplay for the player.

One feature that we would like to improve in the future is the reward system. A reward system provides the player with various incentives to get the player interested and more involved. Ideally, after the player clears a region (or certain number of levels), the player would receive new units of their choice that have unique abilities helpful for defeating enemies. Also, after every few levels, the player would get to choose a small reward from a set of random rewards from the list of rewards. In the future, the upgrades would be able to do more, such as actually evolving units and adding extra abilities. Furthermore, along with functionality based on regions in the game field, there would be different maps and themes for levels, adding variations to the game. The map would include different enemies and challenges specific for each region. Accordingly the player must factor into their strategies to progress. This will also increase the difficulty and fun of gameplay.

Funding

This work has been supported by the Organized Research and Creative Activities (ORCA) Program of University of Houston-Downtown.

Conflict of Interest: none declared.

References

Adams, Ernest. (2014) Fundamentals of Game Design (3rd edition). Publisher: New Riders.

Avery, Phillipa & Togelius, Julian & Alistar, Elvis & Leeuwen, Robert. (2011). Computational Intelligence and Tower Defence Games. 2011 IEEE Congress of Evolutionary Computation, CEC 2011. 1084 - 1091. 10.1109/CEC. 2011. 5949738.

Banks, Alec; Vincent, Jonathan; Anyakoha, Chukwudi (2007). "A review of particle swarm optimization. Part I: background and development". Natural Computing. 6 (4): 467–484.

Cooper, K. (Ed.), Scacchi, W. (Ed.). (2015). Computer Games and Software Engineering. New York: Chapman and Hall/CRC.

Eichenbaum, A., Bavelier, D., Green C. S. (2014). Video Games: Play That Can Do Serious Good. American Journal of Play. Volume 7, number 1.

ESA, 2019 Essential Facts About the Computer and Video Game Industry. https://www.theesa.com/esa-research/2019-essential-facts-about-the-computer-and-video-game-industry/

Ferrara, John (2011). The Elements of Player Experience. UX Magazine, Article No:651.

Miikkulainen, R. (2006). Constructing Intelligent Agents in Games. Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2006 Symposium.

Nara, R. (2012). Benefits of mixed flocks. *Nature* **492,** 314 (2012) doi: 10.1038/492314a

Reas, Casey and Fry, Ben (2014). Processing: A Programming Handbook for Visual Designers, Second Edition. MIT Press. ISBN: 0-262-02828-X

Reynolds, Craig (1987). Flocks, herds and schools: A distributed behavioral model. SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. Association for Computing Machinery. pp. 25–34.

Russell, Stuart J.; Norvig, Peter (2009). Artificial Intelligence: A Modern Approach (3rd ed.). Upper Saddle River, New Jersey: Prentice Hall. ISBN 978-0-13-604259-4.

Woolf, Beverly & Lane, H. & Chaudhri, Vinay & Kolodner, Janet. (2013). AI Grand Challenges for Education. AI Magazine. 34. 66-84.