Design Patterns Applied for Game Design Patterns

Junfeng Qu[1], Yinglei Song[2] Yong Wei[3],
[1] Department of Computer Science & Information Technology, Clayton State University, Morrow, GA, 30260
[2] Department of Computer Science, Jiangsu University of Science and Technology, Zhenjiang, China, 212001
[3]Department of Computer Science, University of North Georgia, Dahlonega, GA 30597

Abstract — **We analyzed common game design patterns such as enemy, weapon, communication, and creative control as well as social interaction and explored design patterns that are applicable to these game design patterns. The reusability, maintainability, and quality of game development are supposed to be improved and game development is able to adapt to constant changing and evolutionary game building process. Although some design patterns are discussed, other design patterns might suitable as well because game programming is too complicated to separate and apply each pattern independently.**

Keywords –Video Game, Game Design Patterns, Design Patterns, Communication, Interaction

I. INTRODUCTION

A. Video Game and Game Development

Video game was called interactive electronic game in late 1940s. Alan Turing and Champernowne wrote a chess-playing algorithm in 1948. In 1950s and 1960s, The video game is still in the stage of interactive electronic game, although Tennis for Two, Spacewar!, and Space Travel were published during that time. Between 1970s and 1980s, computer has been developed and programming has developed better functional decomposition to build overall logic of the program, video games were also moved to computer video games stage although these games, such as Start Riders (1979) Donkey Kong (1981) etc., are simple arcade, beeping, or simple consoles.

By the end, the video game has moved onto character-based, multi-genre entertainment monster such Super Mario Bros, SimCity in the late 1980s. Object-oriented programming surfaced during the 1980s. The OOP enables game developers to use multiple classes, and be able to refine operations within classes and relationships between classes. During 1990s, design patterns extend the object-oriented paradigm and enables programmers to start programming tasks with a component view of their work.

Although game developers has a better tools to develop game. 2D and 3D games around 1990s overall are still very simple in terms of project size and complexity of requirements and functions[1]. In general, these games components include sound, File I/O streaming, Main, Simulation, 2D or 3D graphics and rendering, simulation, collision detection.

In Past ten years, games have ballooned in complexity. Video game is not only run on console, PC, but also on mobile devices, or over networked game server, such as Halo 5, Dungens and Dragons. The video game development involves in not only 2D or 3D graphics, I/O, simulation, but also involves persistent data store, network scene management, server gameplay, AI, world construction and layout, scripts processing etc. Now the primary technical challenge is simply getting the code to work to produce an end result that bears some balance to the desired functionality[2]. The problems are mainly due to overall project size and complexity and highly domain-specific requirements.

Abio etc.[3] found that most cited problems are the unrealistic or ambitious scope and features creep with 75% of the project complaining followed by the cutting of features during development phase. In game development, the problems in the design phase and delays or too optimistic project schedule, technical problems are also exists in most game development process. All the main problems of the traditional software development problems that exist in software industries are also found in the game industry as well.

Overall, game development requires software engineering methodology to adapt to the game industry and covers aspects such as complexity, multidisciplinary collaboration, creativities of game designer and artist. A well designed game programming, process, and architecture should be modulated and integrated with software development procedure well and be able to extend easily, portal to other platform easily without deep revision of source code, and be able to transform to other game platforms with minimized transform time and efforts.

B. Design Patterns in Game Development

There are mainly two issues in game development, one is how to solve the issues with game design, which deals with versatile game design related issues such as building blocks of games, game world rendering, character design, level design etc. The other is how to implement and deliver game designed via programming language with minimal efforts and adapt to constant changing game development process. Game design patterns were first introduced by Kreimeier[4] in 2002. Then Björk et al. studied how players interact with games and how entities in a game interact with each other [5]. They focus on reoccurring interaction schemes relevant to game's story and core mechanics of game. After interviewed with professional game programmers, the authors analyzed the existing games and game mechanics and then proposed those more than two hundreds game design patterns that describe elements of and approaches to video game that can be found regularly. The patterns are organized in broad categories and described in terms of how they are used the choices a designer must make when using them, their consequences and relationships to other patterns.

Lewis[6], Hullett[7], Milam[8], Lankoski[9] etc have explored video game design patterns further in the area of game dialog, level design, player's movement, NPC behaviors etc. Dahlskog et al.[10] analyzed search-based procedural dungeon generator patterns.

Another design patterns are proposed in the field of computer science by GoF[11]. These design patterns guide the perceptions of the designer's approach to design and improve the quality of a program. Design patterns are proven solutions to well-established software engineering problems. In game programming, programmers are often tend to crunch the correctness of a program by evaluating its behavior of character, and overlook the design aspect, such as open-close principle, scalability, maintainability, flexibility, extensibility, and robustness to changes. Therefore, programmer often has to rework or completely discard their work complete in order to accommodate changes of algorithm, evolution of game level, or a changed game mechanics that render player and game interaction during the game development process.

Researchers in computer science have pioneered to teach design patterns in computer programming and object oriented design classes[12][13][14][15][16][17]. Most of these research focuses on explain design patterns using a game , and show how effectively there are represented in case studies, such as computer game[13], the Game of Life[16], the Game of Set[14] and [15]. It's have been shown that computer games provide a context that is familiar and motivating to students, use games as a series of case studies, the students were able to learn better than each isolated examples.

Ampatzoglou and Chatzigeorgios[18] had evaluated the usage of design patterns in game development. Games development is often a process of quick evolution, demands great flexibility, code reusability and low maintenance cost. Games differentiate themselves between versions greatly in terms of characters, terrains, game mechanics and game play. Based on the reverse engineering technology, the authors studied two open-source games and found that design patterns can be beneficial with respect to maintainability. The game version that uses patterns has reduced complexity and coupling compared to a prior version without patterns. The game cohesion is also increased with design patterns applied in game development. It concludes that if design patterns are used properly and in appropriate cases, the programming maintainability, extensibility, flexibility and comprehensibility can be extremely beneficial and improved.

Dung Nguyen and Stephen Wong[19] had applied MVC design pattern to the overall architecture of the board game Tic-Tac-Toe programming. Narsoo etc.[20] demonstrated the used of design patterns to develop game and mobile devices. Game Memento pattern, MVC, game state observer pattern, and change screen patterns are studied with Sudoku game. They all concluded that developers could write better re-usable code faster and allow modification to be made without opening up too many classes. Applying of design pattern facilitated the design of game.

Chowdhury etc. [21] studied applying design patterns to enable auto dynamic difficulty in video games with more reusability and lower risk compared to traditional ad hoc approach. Chen [22] addressed portability issues in game development with MVC pattern, which allow the same game to be ported to different STB environments without or with a little modifying the game.

We[23][24] have studied applying design patterns in game programing to handle expressive play, game state management, scene rendering, and character rendering to increase reusability, maintainability, and quality of game development to adapt to evolutionary game development process.

In this paper, we will focus on using design patterns to accomplish some typical game design patterns discussed by Hullett, Giusti and Björk. The paper is organized as follows: Section II exams three game design patterns. Section III illustrates possible applications of design patterns to address these game design patterns that are mostly seen in games, and Section IV is the summary of the research.

II. Related Game Design Patterns

Modern games involve avatars, one or more ranged weapons, and a varying number of enemies, such as Halo, Call of Duty, Battlefield, Killzone etc. Action and shooter game almost take half of the video game business. Rivera[25] etc. studies Non-player Characters(NPCs) design patterns in shooter games on how to use NPCs, behaviors, types to enabling more engaging experiences. Giusti etc.[26] further explored weapon design patterns in shooter game on classification weapons based on the physical inner-workings of the weapons, behaviors, and how weapon can be used to influence gameplay.

A. NPCs Design Patterns in Shooter Game Design

The work of Rivera[25] defines the elements of a NPC includes following:

Movement Type – describes the way the NPC moves in a combat situation.

Movement Range – describes how far the NPC will move during an engagement

Movement Frequency – defines how often the NPC will change position during an engagement.

Attach Frequency – describes how often the NPC will initiate an attach

Weapon Type – define category utilizes the weapon patterns (Snipping, Close Blast, Assault Weapon, Projectile, Power Weapon, Melee Weapon)

Weapon Damage – defines how much damage the NPC will do to the player's health, Shield, or Armor.

Armor/Health – denotes how much damage the NPC can take before being killed.

Motive – indicates of what type of combat encounter the NPC would create and shows its purpose to the design. Three main factors that an NPC can affect:

- Challenge – the degree of difficulty within a combat encounter
- Tension – The degree of mental stress the player experiences during a combat encounter
- Pacing – the degree of movement that the player will engage in during a combat encounter.

Based on the elements of NPC, the following patterns and sub-patterns in non-player character design are defined:

- Soldier – an NPC that pressures the player from range
  - Grunt – a weak NPC that attacks from a medium distance, often in groups
  - Elite – a strong NPC that works from a medium distance
  - Grenadier – A weaker NPC that maintains a long distance to encourage players to move forwards
  - Sniper – an NPC that cause high damage from a long distance
- Aggressive – An NPC that attempts to close the distance between itself and its target
  - Suicidal – rush at the player at the cost of its life
  - Swarm - rushes the players in group, but individual damage is low
  - Berserker – a strong NPC that cause a high amount of damage over a prolonged amount of time
- Carrier – An NPC that spawn more NPCs during an encounter
  - Sacrificial – an NPC creates more NPCs once it dead
  - Summoner – spawns more NPCs at a distance
- Tank – an NPC that poses a significant singular threat and prevents the player from proceeding.
  - Turret – a slow-moving NPC that cause high damage at a long range
  - Shielded – an NPC with a large amount of armor, but only in a single direction

B. *Weapon Design Patterns in Shooter Game Design*

Giusti[26] showed that weapon patterns considers how much damage the weapon deals, the range, the area effect, cooldown, capacity, special effects etc. Six weapon design patterns are studies in the research as following:

- Sniping Weapon – engages from a long distance and do large amounts of damage per shot. Sometimes with vision magnification system and aim accurately. Sniper NPC sometimes Turrets and Elites use this type of weapon
- Close Blast – fires in a quick and inaccurate manner, such as Shotguns, flamethrowers. Berserker and Elite NPC often use close blast to catch the player in close-quarters tactically
- Assault Weapon – fires accurately and quickly in mid-range, but the damage per shot is low. Grunts and Elite NPCs often use this type of weapon to pester the player from a distance although the damage is small
- Projectile – are often explosive objects thrown or fired in a physics-defined arch with damage in a large area of effect. The reload times are long and capacity is often small. Grenadiers, elites, and some time Tanks NPCs use it to force the player out of cover and impose a greater threat
- Power Weapon – gives the player a clear advantage over other available weapons by either being incredibly powerful or by bestowing unique abilities. This type of weapon is often limited by the game designer. Tanks NPC often use this weapon, sometimes Berserkers and Elites use this to threat the player as well
- Melee Weapon – are hand-to-hand weapons such as knives or bare hands. The damage is high and firing-rate is low. Melee weapon often used by player or NPCs at close range with stealth.
- Placed Weapon – is placed in a stationary location and often acts independently with its own logic. The area of effect is medium to large with high damage. Occasionally Elites and Tanks NPC use placed weapon

C. *Creative Control Game Design Pattern*

A good game always catch players' attention and make players focus on gameplay to the extent that they feel immersed in the game. To achieve immersion, game designers often provides means to allow players to express themselves in the game world with unique combination character. In role playing game or online game, it's a cheap way to allow players to express themselves via customizable avatar appearance. The Players have the ability to be creative within the game world [4] is defined as Creative Control. Some games also tie avatar templates that have different capability in the game as well as impact to the outcome and players' path in the game, for example, Dungeons and Dragons provides players with six characterization attributes in terms of Strength, dexterity, intelligence, wisdom, charisma, and constitution. Players are able to customize cosmetic attributes like clothing, hairs etc. as well as to choose from different prototypes of avatar with different functional attributes.

D. *Social Interaction and Communication Channel Game Design Pattern*

Social Interaction is when two or more players have two-way communication between each other, i.e., the other players can respond to the individual player's communication[4]. Players will send or receive message after changes to game state. Players may send information such as triggered events, enemy found, player's status, location etc. to team mate to collaboratively achieve goal. The communication channel is not only for players, but also allow collaborative work of Enemies that try and hinder players' goals as well. For example, once players triggers action of enemies, the enemies, which are often defined by non-player character (NPC), are also need to communicate one-way or two-way to reflect changes of game state because of players' interaction with the game world.

III. Applying Design patterns for game design

One of the unique characteristics of game development and programming is rapidly evolutional, quite often modification and goal changing during game design and development, therefore it's very common that game programmers have to dispose their works that they have been working for months and to restart again. Therefore a well-designed game program would spend minimal efforts and changes to migrate. A well designed game programming that offer great flexibility, code reusability, extensibility, and low maintenance costs is highly desired.

A. Fast Enemies Clone with Prototype Pattern

Video game renders a large amount of similar enemies at different locations to challenge and hinder player to achieve goal easily as well keep players focus on tasks and goals. The capability of these enemies might be different depends on location in the scene or in the team with other enemies. One solution is to create an abstract Enemy class and then different subclasses can be built based on the requirements of power and capability of enemy such as *Soldier, Aggressive, Carrier, Tank* in shooting game as discussed in Section II A. Factory methods can be used here to create different enemy as needed. The problem of this approach is that the number of classes is going to be out of control if the number of capabilities of enemies required in the game increased enormously. On the other hands, the initialization of each type of enemies is time consuming and might not meet the need of rapid shift and change requirements of shooter game.

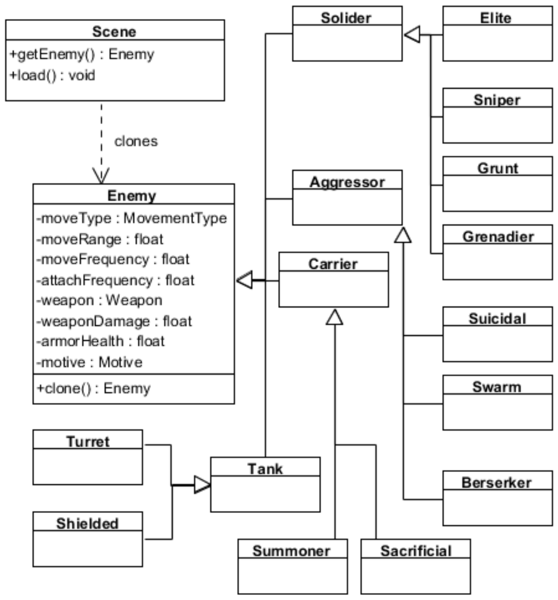

Figure 1. Prototype pattern to clone massive enemies

Prototype pattern is able to create new objects by copying its prototype where the interface for cloning is defined. The subclass implements operations for cloning itself. After the object is initialized, the developer only justify the enemies capability based on the location on the maps. The design is shown in figure 1. The abstract *Enemy* class defined the common attributes such as location, power, speed etc. the inherited classes, e.g. solider enemy, elite enemy implement each concrete operations and clone. The Scene class is responsible to creates new object by asking *Enemy* to clone itself, and adjust slightly each clone based on requirements.

B. Enemy State and Behavior with Strategy and State Patterns

The NPCs in Shooter game are often designed with different state as well such as the normal, the suspended, killed, damaged etc. The character also behaves different in different game plays. The state pattern and strategy pattern can also be used in shooter game to handle different states and different behaviors in game development as shown in figure 2. The *MovementType* is an abstract state class, it's subclasses are concrete classes that implements the transition between different state with *pause(), start(), attacked()* methods etc. The state pattern allow the developer to adapt changes of game design in character development and extend movement type easily with subclasses. According to Rivera etc. the common movement types are

- Flanking Intensive – The NPC move to attack from unexpected directions.
- Passive – no move when attacking
- Slow Push – move slowly to player
- Rush – dash to target
- Cautious – move around and maintain a distance from its target

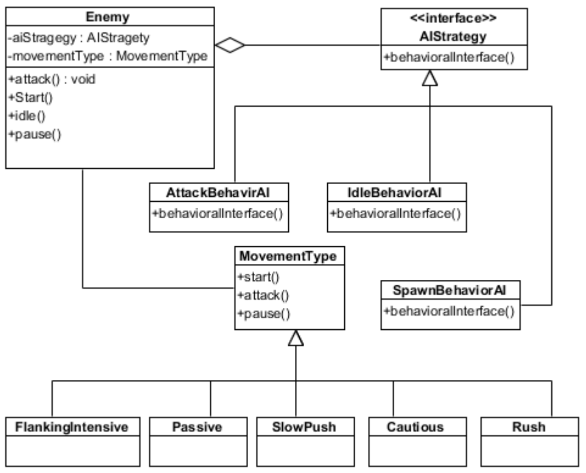

Figure 2. State and Strategy patterns for Enemy

Changes strategies and tactics according to the environments in terms of movement type such as attack, chase, spawn, or wander are abstracted in *AIStragegy* class. These behaviors often changes as the storyline rolled out or scene changes. It is a good practice that developer abstract a basic type of behavior from a given class and place the behavior into separate classes. The abstract class provides an interface for accessing and concrete class execute the varieties interpretations of behaviors such as attach, wander differently as needed. This is how strategy pattern works to adapting evolutions of NPCs' behavioral changes during game design and development. For example, the NPC soldier class dynamically changes strategies as game scene changes and player's inputs changes, the NPC could update behaviors to attack, escape, idle to attract player as a Grenadier etc.

C. Builder Pattern for Creative Control

Video game often allows the player to express themselves via choosing an avatar that reflects the player's personality. Player will be also to customize the avatar such as clothing, gender, body type, skin color, power strength, intelligence etc. These customization can occur both at the beginning of the game and during the play of the game through upgrades awarded or progression of the player.

The steps to create a character are very similar, each component is created, then these components are assembled in to different and complete game character. Builder pattern is a natural solution to create a complex object that contains multiple components.

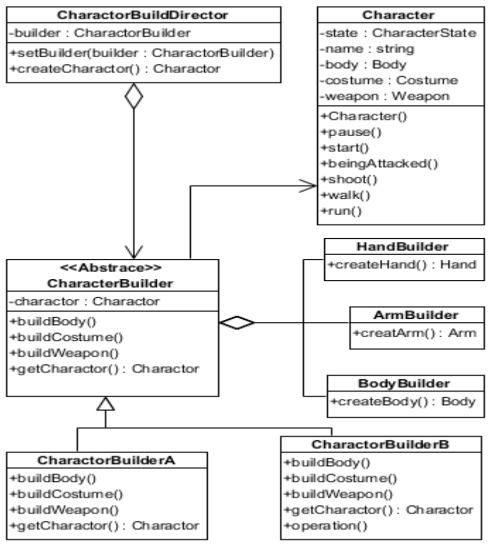

Figure 3. Builder patterns for Creative Control

Figure 3 shows a skeleton builder pattern to create different character that has *Body*, *Costume*, etc. The abstract *CharacterBuilder* class provides interface to create each part of the character, the subclass of *CharacterBuilder* is responsible to build each concrete character with specific customized part directed by *CharacterBuilderDirecter* that knows how to build each concrete character.

D. Dynamic Weapon Functions

In Giusti's weapon game design pattern[26], different sorts of weapons are discussed. To implement these weapons, game designer may dynamically add more functions to these weapons such as power-up, wear-out, repair etc. Game designer may have versatile requirements of weapons, some weapons can wear but not repairable, some never wear, some can't be power-up, some can power-up, wear, and repair. It is more likely that more functions possible modified, added or removed. If general inheritance is used to implement weapon game design pattern, the problems are possible too much level of inheritance, or too many subclasses, and soon it gets out of control of developers.

We introduced factory method pattern that creates different categories of weapons and facilitate extension of new weapons that were not covered in the weapon game design pattern. The decorator pattern provides a flexible way to extend functionality with attached additional functions to weapon dynamically with new functions such as scope, silencer, wear, power-up of different combination dynamically. Figure 4 show the class diagram of the designed application.

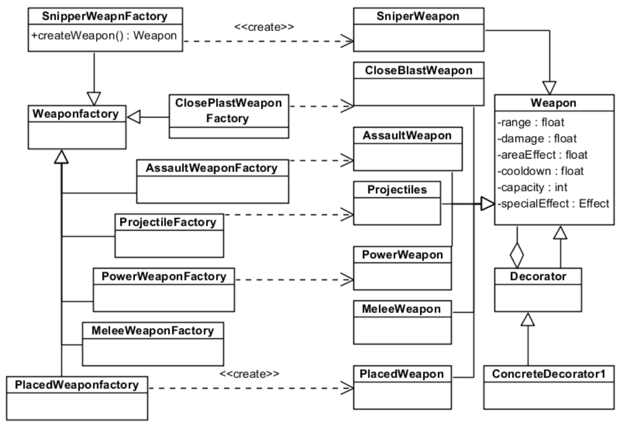

Figure 4. Factory Methods and Decorator Patterns in Weapon Management

In the design, The *WeaponFactory* is an abstract class and can be extended with different concrete class such as *SnipperWeaponFactory* etc. to create different concrete weapons. The *Weapon* class is an abstract and is extended with different concrete weapons to realize different functions. The *Decorator* class maintains a reference to a *Weapon* object and defines an interface that conforms to *Weapon*'s interface, and give developer flexibility to extend functions such as power-up, wear, silencer etc. to meet requirements of game designer.

E. Mediator and CommandPatterns for Social Interaction and Communication Channel

Social interaction or communication channels show how game designer use communication to inform player, goal

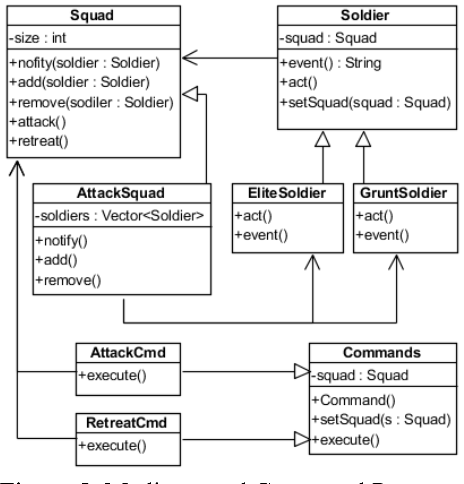

Figure 5: Mediator and Command Patterns

collaboration, and real-time game state change notification. The communication can be one-way or two-way. Use Soldier of NPC as an example, *Soldiers* compose *Squad* that is represented by a mediator, each concrete soldiers such as Elite, Grunt etc. If a soldier's events are triggered such as find enemy, being hit etc., the soldier notifies the Squad, and then the Squad updates all soldiers in the squad and the soldiers act correspondingly.

Every squad acts as a receiver that has a number of different commands, such as different attack or retreat commands. The commands are invoked by event handler to execute different commands while player interacts with game objects. Command and mediator patterns is shown in figure 5.

IV. Conclusion

In this paper, we have applied design patterns to provide solutions for some common game design pattern so game development will adapt to changes of game designer better in terms of software reusability, maintainability, and extensibility.

We applied design patterns to fast enemy render in game world with prototype pattern, enemy behavior and state management with strategy and state pattern, dynamic weapon functions implementation with decorator and factory method, builder pattern for creative control, as well as mediator and command patterns for social interaction and communication. To further evaluate the effectiveness of design patterns in video game design patterns, we plan to conduct a software quality metrics analysis in terms of size, complexity, coupling and cohesion in near future for a specific game.

References

[1] C. Fernández-vara, "The Game Studies Practicum : Applying Situated Learning to Teach Professional Practices 2 . 3 The Difficulties of Teaching Professional Practices," pp. 25–32, 2008.

[2] J. Blow, "Game Development: Harder than you think," *Queue*, vol. 1, no. February, p. 28, 2004.

[3] F. Abio, M. Pimenta, F. Trindade, and C. Dietrich, "What Went Wrong ? A Survey of Problems in Game Development," vol. 7, no. 1, pp. 1–22, 2009.

[4] B. Kreimeier, "The Case For Game Design Patterns," 2002. [Online]. Available: http://www.gamasutra.com/view/feature/132649/the_case_for_game _design_pa tterns.php. [Accessed: 15-Jan-2015].

[5] J. Björk, S., Holopainen, *Patterns in Game Design*. Charles River Media, 2004.

[6] C. Lewis, N. Wardrip-Fruin, and J. Whitehead, "Motivational game design patterns of 'ville games," *Proc. Int. Conf. Found. Digit. Games - FDG '12*, p. 172, 2012.

[7] K. Hullett and J. Whitehead, "Design patterns in FPS levels," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games - FDG '10*, 2010, pp. 78–85.

[8] D. Milam and M. S. El Nasr, "Design Patterns to Guide Player Movement in 3D Games," in *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games - Sandbox '10*, 2010, vol. 1, pp. 37–42.

[9] P. Lankoski and S. Björk, "Gameplay Design Patterns for Believable Non-Player Characters," in *Situated Play, Proceedings of DiGRA 2007 Conference*, 2007, pp. 416–423.

[10] S. Dahlskog, J. Togelius, and S. Björk, "Patterns, dungeons and generators," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*, 1995th ed. MA, 2003.

[12] P. Gestwicki and F. Sun, "Teaching Design Patterns Through Computer Game Development," *ACM Educ. Resour. Comput.*, vol. 8, no. 1, pp. 1–21, 2008.

[13] P. V. Gestwicki, "Computer games as motivation for design patterns," *ACM SIGCSE Bull.*, vol. 39, p. 233, 2007.

[14] S. Hansen, "The Game of Set – An Ideal Example for Introducing Polymorphism and Design Patterns," in *SIGCSE*, 2004, pp. 110–114.

[15] M. A. Gómez-Martín, G. Jiménez-Díaz, and J. Arroyo, "Teaching design patterns using a family of games," *ACM SIGCSE Bull.*, vol. 41, no. 3, p. 268, Aug. 2009.

[16] M. R. Wick, "Teaching Design Patterns in CS1 : a Closed Laboratory Sequence based on the Game of Life," in *SIGCSE*, 2005, pp. 487–491.

[17] A. Rusu, R. Russell, R. Cocco, and S. DiNicolantonio, "Introducing object oriented design patterns through a puzzle-based serious computer game," *2011 Front. Educ. Conf.*, pp. F1H–1–F1H–6, Oct. 2011.

[18] A. Ampatzoglou and A. Chatzigeorgiou, "Evaluation of object-oriented design patterns in game development," *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 445–454, May 2007.

[19] D. ("Zung") Nguyen and S. B. Wong, "Design patterns for games," *ACM SIGCSE Bull.*, vol. 34, no. 1, p. 126, Mar. 2002.

[20] J. Narsoo, M. S. Sunhaloo, and R. Thomas, "The Application of Design Patterns to Develop Games for Mobile Devices using Java 2 Micro Edition.," *J. Object Technol.*, vol. 8, no. 5, p. 153, 2009.

[21] M. I. Chowdhury and M. Katchabaw, "Software design patterns for enabling auto dynamic difficulty in video games," in *17th International Conference on Computer Games (CGAMES)*, 2012, pp. 76–80.

[22] G. Chen, "Digtal STB Game Portability Based on MVC Pattern," *2010 Second World Congr. Softw. Eng.*, pp. 13–16, Dec. 2010.

[23] W. Y. Qu Junfeng, Song Yinglei, "Applying Design Patterns in Game Programming," in *SERP'13*, 2013, pp. 160–166.

[24] J. Qu, Y. Song, and Y. Wei, "Design patterns applied for networked first person shooting game programming," in *2014 IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*.

[25] G. Rivera, K. Hullett, and J. Whitehead, "Enemy NPC design patterns in shooter games," *Proc. First Work. Des. Patterns Games - DPG '12*, pp. 1–8, 2012.

[26] R. Giusti, K. Hullett, and J. Whitehead, "Weapon design patterns in shooter games," in *Proceedings of the First Workshop on Design Patterns in Games - DPG '12*, 2012, pp. 1–7.