American Journal of Advanced Research, 2018, 2–1
December. 2018, pages 7-13
doi: 10.5281/zenodo.1410734
http://www.ibii-us.org/Journals/AJAR/
ISBN 2472-9264 (Online), 2472-9256 (Print)

The Development of a Game with Applications of Object-oriented Programming Concepts

Yanzhi Huang[1], Ting Zhang[2], Ling Xu[3,*],

[1,2,3] Department of Computer Science and Engineering Technology, University of Houston – Downtown, USA.

*Email: xul@uhd.edu

Received on July 30, 2018; revised on August 28, 2018; published on September 1, 2018

Abstract
In this game development project, we explored effective ways of applying object-oriented programming technologies by making a fun and exciting game called TH Tank. TH Tank is a 2D tank shooter game, which includes rich groups of entities with various properties. In this paper, we introduce our design of the game's core-mechanism, entity definitions, the graphical design of game components, and a few innovative features. Our explorations can be helpful to the development of other games and applications which may involve OOP techniques.

Keywords: Game Development, Object-oriented Programming

1 Introduction

Today, with the development of advanced technologies and their applications in games, games possess more and more fantastic and attractive features. Playing games has become a ubiquitous part of many adults and children's lives (Granic et al., 2014). Although many of the people hold negative opinions against games with concerns of the potential harm from game playing including aggression, addiction, and depression (Granic et al., 2014; Ferguson, 2013), the positive effects of games also have received considerable attention from researchers (Gentile and Anderson, 2006; Baranowski et al., 2013; Thompson, 2012). Games not only can be used for entertainment purposes, but also contribute to computer science teaching and learning (Cooper and Scacchi, 2015). Learning game development can greatly motivate student learning interest and promote their critical thinking skills (Kosa, 2016)). In this paper, we discuss the development of a game, with explorations of object-oriented programming (OOP) concepts, which are important but usually challenging to computer science students in their university study.

Object-oriented programming is a programming paradigm based on the idea of a collection of interacting objects (Dathan and Ramnath, 2015). A remarkable feature of OOP is its facilities to "the separation of concerns", a phase described by Dijkstra (Dijkstra, 1982) in 1974. The separation of concerns can be interpreted as separating a program into distinct objects, focused on the functionalities from their own points of view (Glasser, 2009). Through the separation of concerns, we can encapsulate the knowledge of an object for the purpose of access authorization and protection. In addition, the modularization via the objects makes possible the components in one program being reused in other programs. Due to

Figure 1. Upper: the game menu; lower: a screen shot of the game demo.
the above beneficial features, OOP has been widely used in many applications. Here we present an exploration of OOP techniques via game development. Developing a video game usually requires knowledge from various aspects, such as computer graphics, software engineering, artificial intelligence, and networks. Here we focus on introducing an application of OOP concepts in developing a shooter game. Our game has entities (such as tanks, bullets, and timer) demonstrating distinct properties. Those entities are defined as classes that encapsulate the attributes such as color, and functionalities such as moving. There are also entities, such as different tanks, sharing some common properties (such as appearance) but different in other properties (such as power) and behaviors (such as attacking and chasing mode). They are implemented by applying two OOP concepts: inheritance and polymorphism. In addition to basic functions of an action game, our game also possesses innovative features by combining some strategy-based game elements. A few random elements are included in the core mechanics, for uncertainties and fun of the game play. Figure 1 shows two screenshots of our game demo.

Our work makes the following contributions. First, we successfully applied the concepts and methodology of OOP in the development of an action game; the experience can be helpful to the development of other games and the applications which involve OOP techniques. Second, we combined features from different game genres, which is a valuable exploration on game development ideas. Third, we introduce a game prototyping and implementation environment – Processing. It is effective in teaching and studying game programming without a complicated user interface and tedious parameter settings, but friendly to novice users. It is worthwhile to mention that all the implementation work was done by an undergraduate CS student, who initially had no OOP knowledge, with instructions by a mentor. During the development process, we explored effective methods for teaching and studying OOP, by relating the abstract concepts to solid game entities and interactions. Although this is a single case exploration, we believe it provides a good start and prepares for our further work especially OOP teaching in computer science or programming training in game industries. The rest of the paper is organized as follows. Following the introduction, the section of background will introduce object-oriented programming concepts, related knowledge about games and game development, and the game development tool that we use. The details of our implementation work will be introduced in the game design section. The last section includes a conclusion and also proposes the future work.

2 Background

2.1 Motivation

The motivation of this project comes from the authors' experience in teaching and learning game programming where object-oriented programming concepts cannot be avoided. Students often show high enthusiasm in game playing but feel frustrated in game programming. The difficulties exist in the understanding of the abstract OOP concepts and how to relate them to specific entities existing in the game world. A common problem that we observed from class (according to the author's teaching experience in C++, Python, game programming, and object-oriented programming with Java) is the lack of object-oriented thinking. It is also a problem previously mentioned by Beck and Cunningham (Beck and Cunningham, 1989). They pointed out that "the most effective way of teaching the idiomatic way of thinking with objects is to immerse the learner in the 'object-ness' of the material", We believe the entities in a game world can make "'object-ness' of the material" concrete and resonant to university students, since everything in a game world is an object such as the monsters, the coins, the obstacles, etc. For most of students, they have the experience of game playing and can naturally understand the objects and how they interact in an object-oriented way.

2.2 Related work

The game we decide to develop is a 2D shooter game with a top-down perspective, due to the simplicity for novice game programmers. In addition, it possess most of key game elements, including various entities such as enemies and bullets, score system, timer, and collision detection mechanism. The most important aspect that we care about is that the implementation of the above elements will be initiated from the developer's object-oriented thinking, which is a valuable training to OOP study.

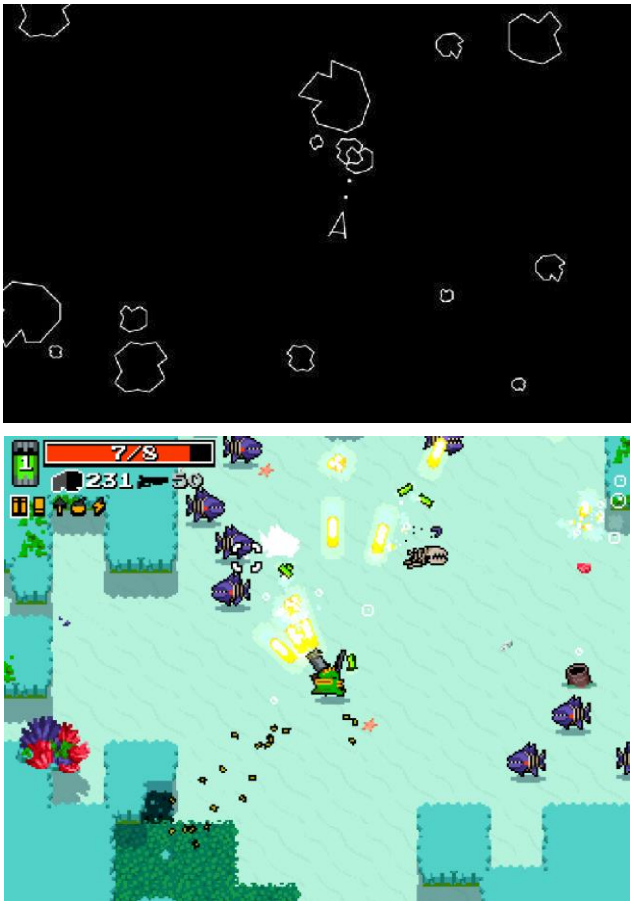

Figure 2. Upper: the game *Asteroids*; lower: the shooter game *Nuclear Throne*. Both games are with a top-down perspective.

It is well known that shooter games are a subgenre of action games, one of the most popular game genres that are familiar to many people. A milestone of the early shooter games is *Asteroids* (as shown in the upper of Figure 2) released in 1979 (Salen and Zimmerman, 2004). The objective of the game is to destroy asteroids and saucers by firing shots from a player-controlled triangular spaceship. The spaceship can rotate left and right with the player's key presses. The player's score increases as asteroids and saucers are destroyed, and decreases as they hit the spaceship.

The game includes a few levels: once the screen is cleared with asteroids and saucers, the next level starts with more challenges. Even the early asteroids game does not have fancy visual effects, it includes the design of variety and randomness for fun, which is an important feature in modern games (Bjork and Holopainen, 2005). For example, a big asteroid can break into a few small asteroids when it is shot; the small ones move faster and worth more points. The saucers also demonstrate different properties and random behaviors. The above design helps to keep the gameplay tension and player's fading interest when the game playing time progresses (Adams, 2009). Another example of the later shooter games is *Nuclear Throne* released in 2015 (Devore, 2015). Compared with the game *Asteroids*, it includes more various entities – totally 12 characters, each possessing unique abilities. Similar to *Asteroids*, it also follows a linear level structure, which requires the player to progress level by level until the game over. Undoubtedly *Nuclear Throne* possesses more interesting features, such as visual and sound effects, entity behaviors, and the design of secrete characters. In the above two game examples, we are interested in some features, such as the design of randomness, variety of entities, and some fancy visual effects such as explosion and fading. The most important reason for our decision of making a shooter game similar to the above two examples is that, they are actually simplified versions of larger games that possess necessary components including rendering, physics, artificial intelligence, user control of a character, and non-player characters. We incorporate the features in our game design, and also add novel features from other game genres, in order to make the game more interesting. The details are introduced in section 3.

2.3 Development Tool

The tool that we use for game development is *Processing* (as shown in Figure 3). *Processing* is an open source computer language and integrated development environment. *Processing* can be "used by students, artists, designers, architects, and researchers for learning, prototyping, and production" (Reas and Fry, 2014).

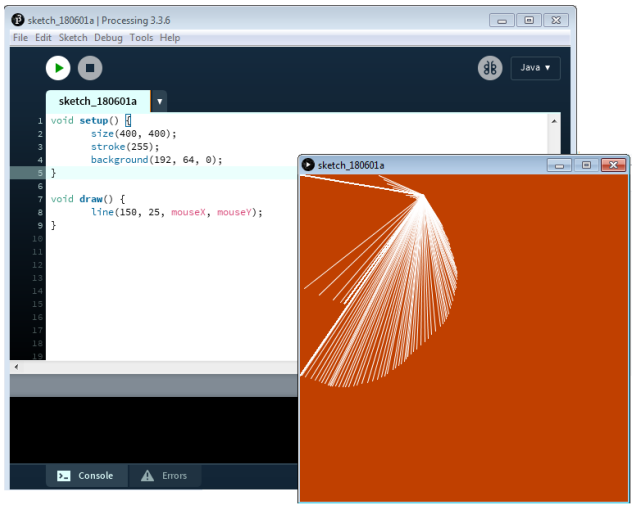

Figure 3. The *Processing* IDE and the window that shows the running result. The program will draw a line in a window from a fixed point to the mouse position dynamically.

We chose *Processing* as our programming tool is because it builds on the Java language but with a simplified syntax, which helps students to focus on the object-oriented thinking and game design rather than the language syntax. Compared with *Processing*'s language, other game programming tools such as XNA and Unity have higher requirement on the game developer's programming skills, which could be an obstacle for novice game programmers such as students. In addition, *Processing* is not focused on handling complicated graphical features - most of the graphics produced in *Processing* is done by CPU instead of graphic card. Codes to render images are much shorter and simpler in *Processing*, compared with some industrialized graphic IDEs such as OpenGL. This feature allows developers to fully focus on game design and programming aspects other than rendering techniques. We also noticed other elementary alternative tools such as scratch (Wolz, 2009) for game development. However, we agree with Alfredo et al.'s (Alfredo et al., 2017) opinion – there exists a few problems for using Scratch in teaching an introductory programming language for a CS1 course in a videogames major. A typical problem is the lack of necessary programming training due to many manual manipulations.

3 Game Design and Implementation

3.1 Object-oriented thinking and implementation

Object-oriented thinking in our game project involves understanding the entities in the game world, their attributes and behaviors, and the interactions between each other. In the implementation level, we define the above using the OOP concepts including abstraction, encapsulation, inheritance, and polymorphism.

Abstract is a key concept of object-oriented programming, while the intention is to reduce the programming complexity by hiding unnecessary implementation details (Dathan and Ramnath, 2015). Abstract is accomplished with definitions of classes and their behaviors. In our game, we create a few abstract object classes and their instances. In each class, we specify the properties or behaviors. For example, we define a class *Bullet* with a property of position coordinates and behavior of updating its position with time.

Encapsulation is packing of the data and actions into a single package (i.e. a class) and via this kind of data hiding can protect the data from outside misuse or interference (Savitch, 2003). In our case, we interpret encapsulation in terms of how the methods of an object are hidden from other objects. For instance, the movement of the player tank is not known to any enemy tank instance.

Inheritance is the mechanism that allows using of an existing class for defining new classes; the new classes (i.e. subclasses) can inherit/share a set of attributes and methods of the existing class (i.e. the parent class). In our game, the general *Tank* class can be used to create specialized classes representing various kinds of tanks such as *PlayerTank* and *EnemyTank*, and each subclass inherits the properties (such as position and health) and methods (such as update of position and collision detection) from the superclass *Tank*.

Polymorphism allows an operation to take different forms, based on the object types. To be specific, in our case, the behavior of shooting bullets differs in different types of tanks: regular tanks can shoot a regular bullet but have to wait for some time in order to cool down and shoot the next. Boss tanks have high possibilities to shoot a power bullet, based on some randomness settings. It is natural to define the function *shootBullets* differently for various tank classes.

In Table 1, we list major entities and their properties. More detailed implementation of these entities and application of the above discussed OOP features is shown in Figure 4.

Table 1. Description of Entities

Entity	Description
Tank	A tank has its center position, and four description points. Those point together determine tank's location and contour, which will be use in collision detection and rendering. Other properties include orientation, speed, status, health, bullet loading time, aiming time, and visual appearance.
Player Tank	It is a child class of the class *Tank*. It inherits all attributes from the Tank class.
Enemy Tank	It is a child class of *Tank*. It inherits all attributes from the Tank class. There are three types of enemy tanks based on their powers/weapons: regular weapon, laser weapon, and the boss tank. They appear with the progression of levels.
Bullet	Every bullet has its speed, position, and the damage value it may cause.
Obstacle	The static obstacles are buildings. They have positions, appearances, and life durations.

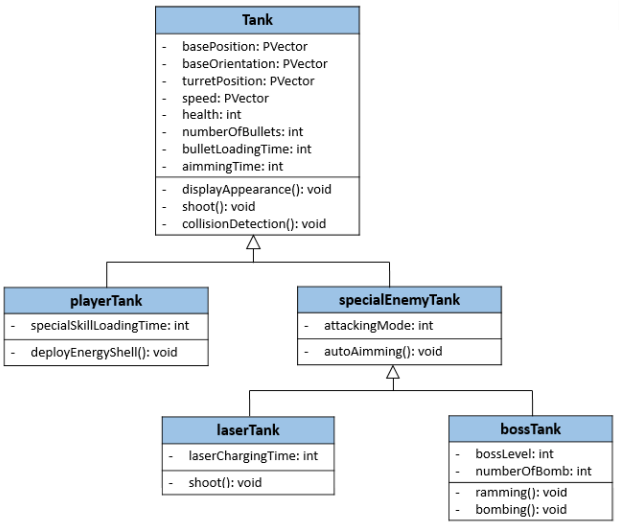

Figure 4. UML diagram of the tank classes.

3.2 Game rewarding

Our game does not involve a complicated story but has a linear progression – the player will progress to the next level once the short term goal in each level is achieved, and finally achieve the long-term goal to destroy the final boss tank. The short term goals can be to destroy a number of enemy tanks, to earn a target score, or to avoid being hit by enemy tanks within limited time. The long-term goal can also be set based on the ideas of the short term goals. For shooter games, an important design principle is that "a player who shoots precisely should do better than one who misses a lot" (Adams, 2009). Based on this principle, our game is designed to reward the player's performance with respect to the hit ratio, the number of bullets, and the time used. The main reason of using the linear progression layout is for the simplicity: we don't want to spend much time in the story design stage but on the OOP programming part. For novice student programmers, other more sophisticated layout such as parallel layout (where a variety of paths in lower levels can lead to an upper level) is more challenging and prone to cause frustration.

3.3 Game AI

As we finish the work to identify the entities and their static properties as stated in section 3.1, we focus on the behaviors of the entities here especially the enemy tanks. Their moving modes are illustrated in the finite state machine (FSM) shown in Figure 5. Finite state machine is a model of computation that can be used to represent a few states of the agent in a game and their transitions under certain conditions. FSM is widely used in computer game AI design, due to its simplicity and ease for prototyping.

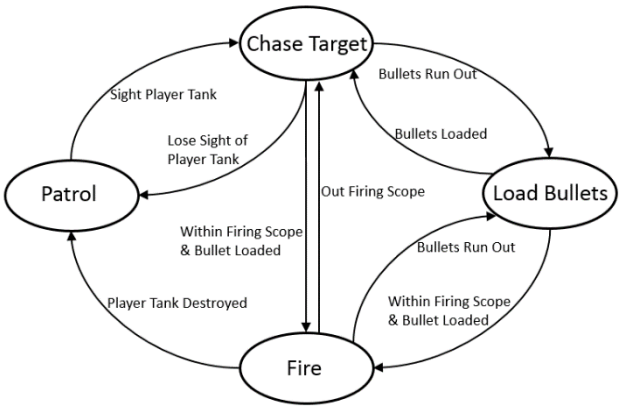

Figure 5. Finite state machine representing the enemy tank's behaviors

As shown in Figure 5, the enemy tank has four states (represented as nodes in the graph): patrol, chase, load bullets, and fire. Invoked by specific events, the entity can transit from a state to another, where transitions between states are represented as edges connecting the nodes. In our case, the default moving mode of the enemy tank is patrolling. Once the enemy tank sees the player tank, it will switch to the chasing mode – it will adjust its moving direction and chase the player tank. If it has bullets loaded, it turns its turret to fire at the player tank; otherwise, it loads bullets first and then fire. The enemy tank will stop firing until the player tank is destroyed or moves out of its firing scope.

In the above process, the events that invoke the state transitions are detected at every game frame. As one of the most important events in the game world, collisions in our game take place between the player tank and enemy tanks, the tanks and obstacles, and the tanks and bullets. The collision detection is based on the distance between two objects: if the smallest distance is less than a threshold, a collision takes place and invokes corresponding effects such as sparks and sounds. Here the smallest distance is the minimum of the distances between the registration/key points (centers of red circles and/or green circles in Figure 6 - 9) on each object. To reduce the computing cost, we can use a circle shell

centered at each object for a rough distance detection. Only when an approximated collision happens, we do precise computations of key point collisions. Figure 6 - 9 shows collisions between two tanks, a tank and an obstacle (building), and tanks and shots, respectively.

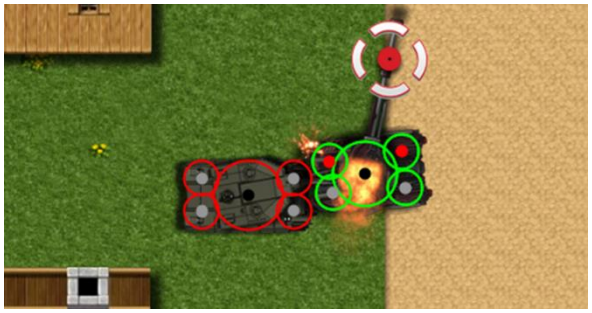

Figure 6. Collision that take place between two tanks. An animation of explosion on a damaged tank is invoked

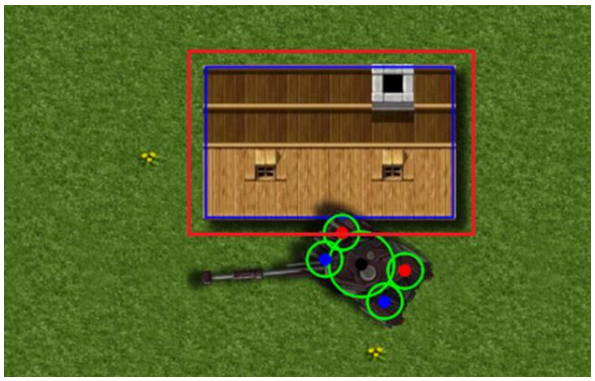

Figure 7. A tank collides a building at a key point.

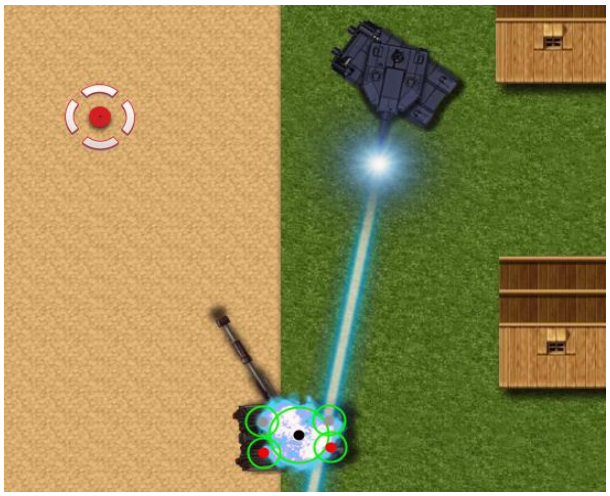

Figure 8. A collision between bullets and tanks.

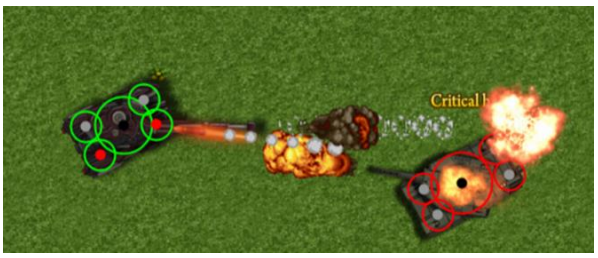

Figure 9. A tank is hit by the laser beam.

In addition to the collision between regular bullets and tanks, there is another type of collision takes place when a special weapon, laser beam, is used. The enemy tank equipped with the laser weapon can fire a beam, and the player tank will get damaged when any point is hit by the beam, as illustrated in Figure 9. Although this collision appears different from other collisions, the detection method is still the same, but with a different visual effect invoked.

In order to increase the game varieties and add fun to the game play, we involve random elements in this game. For example, when the player or an enemy tank is hit by a regular bullet, there is a 50% of chance it will be set on fire, and the fire will act like a debuff which will slowly eat away HP at a rate of 2 points per second. Another example is the variation of tank motion patterns. Some enemy tanks do not follow a constant pattern of motion - they demonstrate different motion patterns each time when you start the game. The purpose of this feature is to make enemy less predictable therefore adding more uncertainty and fun to the game. Player can play this game repeatedly and getting different fights and results every time.

3.4 Visual effects

In this game we have designed a few visual effects. Some effects are associated with special functions of tanks, such as energy shield and ice bullet. Energy shield is designed to protect the player tank from enemy attack. It can be deployed with a user key press and will last for a period of protection time. It can be invoked again after it expires for a while when cooled down. This design is also applied to the ice bullets. Ice bullets are special ammo that can cause an enemy tank frozen for a few seconds. Similar to energy shell, the ability to fire an ice bullets can only last for short time and will recover later. In the game window the status of these two special functions are visualized as state bars.

Figure 10. The effect of an energy shied protecting a tank from shots.

Figure 10 shows a tank protected by the blue energy shield from shots. The top bar shows the recovery time for the next invoke of the energy shield, and the lower bar shows the health status of the player tank. Figure 11 shows the effect of ice bullets. A tank is frozen and loses the ability of moving and firing when hit by an ice bullet. It can resume its

regular status when the effect of ice bullet expires. The upper bar shows the recovery time to invoke the next ice bullet. The plain red bar just below the ice block, i.e. the hit enemy tank, shows its current health status. The design of ice bullet is actually inspired from some RPG games, where the avatar of the player may be frozen due to poison or harm of weapons. To our knowledge, we have never found the frozen function in previous shooter games. In this game, we build this function, and we believe some features in other genres of games can also contribute to shooter games if incorporated with proper modifications.

Figure 11. An enemy tank is hit and frozen by an ice bullet fired from the player tank.

Figure 12. Visual effects for bullet firing and hit.

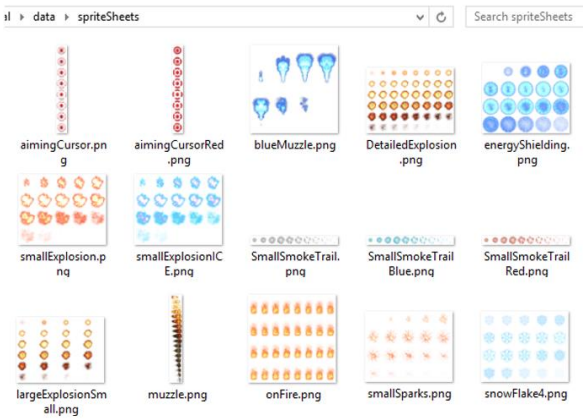

Figure 13. Sprite sheets used for visual effect animations.

In addition to the above visual effects, there are effects invoked by bullet firing and hit, such as muzzle, fire, and smoke trail, as shown in Figure 12. To implement these visual effects, we create a muzzle object and allow it to animate. It is placed in front of the tank turret. In a similar way, for the fire effect, we create a fire object and add it on top of the turret and display the fire burning effect for 10 seconds and then fades off. In order to simulate the trajectories of a bullet flying in the air, we create a smoke object and animate it. To implement the animations of the above objects, we define a function *updateAppearance()* in each class for updating its appearance. At every game frame a sprite sheet is used, from which the function *updateAppearance()* will fetch a small image. The sprite sheets are shown in Figure 13.

4 Conclusion and Future Work

Learning object-oriented programming concepts is often a challenge to CS students. How to teach these concepts in an effective way is also a question for instructors. In this paper we introduce our experience of teaching and learning OOP via a game development project. The OOP concepts are associated with game objects and behaviors, making the learning and teaching process effective and fun. In our future work, one direction is to emphasize the training in OOP concepts using specific game modules. Developing other genres of games is also possible. We expect to explore more factors that affect the OOP study, and build a more systematic method and apply it to undergraduate computer science courses such as object-oriented programming or programming methodology.

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

Alfredo Martínez-Valdés, José & Velázquez-Iturbide, J. Ángel & Neira, Raquel. (2017). A (Relatively) Unsatisfactory Experience of Use of Scratch in CS1. 1-7. 10.1145/3144826.3145356.

Baranowski, T., Buday, R., Thompson, D., Lyons, E. J., Lu, A. S., & Baranowski, J. (2013). Developing Games for Health Behavior Change: Getting Started. Games For Health Journal, 2(4), 183–190. http://doi.org/10.1089/g4h.2013.0048

Beck, K. and Cunningham, W. (1989). A laboratory for teaching object oriented thinking. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '89). ACM, New York, NY, USA, 1-6.

Bjork, Staffan & Holopainen, Jussi (2005). Patterns in Game Design. Charles River Media. p. 60. ISBN 1-58450-354-8.

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

Dathan, B. and Ramnath, S. (2015). Object-Oriented Analysis, Design and Implementation - An Integrated Approach. Springer.

Devore, Jordan (2015). Review: Nuclear Throne. Destructoid. December 27, 2015

Dijkstra, E. W. (1982). Selected Writings on Computing: A Personal Perspective. Springer.

Ernest Adams. 2009. Fundamentals of Game Design (2nd ed.). New Riders Publishing, Thousand Oaks, CA, USA.

Ferguson, C. J. (2013). Violent video games and the Supreme Court. American Psychologist, 68, 57–74. doi:10.1037/a0030597

Gentile, D.A. & Anderson, C. A. (2006). Video games. In N.J. Salkind (Ed.), Encyclopedia of Human Development (Vol 3, pp. 1303-1307).

Glasser, M.(2009). Open Verification Methodology Cookbook. Springer.

Granic, I., Lobel, A., & Engels, R. C. M. E. (2014). The benefits of playing video games. American Psychologist, 69(1), 66-78.

Kosa, M., Yilmaz, M., O'Connor, R., & Clarke, P.M. (2016). Software Engineering Education and Games: A Systematic Literature Review. J. UCS, 22, 1558-1574.

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

Salen, Katie & Zimmerman, Eric (2004). Rules of Play: Game Design Fundamentals. MIT Press. ISBN 0-262-24045-9.

Savitch, Walter (2003). Absolute Java. Pearson Addison Wesley.

Thompson, D. (2012). Designing Serious Video Games for Health Behavior Change: Current Status and Future Directions. Journal of Diabetes Science and Technology, 6(4), 807–811.

Wolz, Ursula & H. Leitner, Henry & Malan, David & Maloney, John. (2009). Starting with scratch in CS 1. ACM SIGCSE Bulletin. 41. 2-3. 10.1145/1508865.1508869.