ISSN: 2508-7894 © 2021 KODISA & KJAI.
KJAI website: http://www.kjai.or.kr
doi: http://dx.doi.org/10.24225/kjai.2021.9.2.35

A Study on 2D Character Response of Speed Method Using Unity*

Dong-Hun HAN[1], Jeong-Hyun CHOI[2], Myung-Jae LIM[3]

Received: October 12, 2021. Revised: November 18, 2021. Accepted: December 05, 2021.

Abstract

In this paper, many game companies seek better optimization and easy-to-apply logic to prolong the game's lifespan and provide a better game environment for users. Therefore, research will be showing the game's key input response method called RoS (Response of Speed). The purpose of the method is to simultaneously perform various motions with the character showing natural motion without errors even if the character's control key is duplicated. This method is for the developers so they can reduce bugs and development time in future game development. To be used with quickly generating game environments, the new method compares with the popular motion method, so which method is faster and can adapt to diverse games. The paper suggested that the Response of Speed method is a better method for optimizing frames and reducing the number of reacting seconds by showing a faster response and speed). With the method popularity of scrollers, many 2D cross-scroll games follow the formula of Dash, Shoot, Walk, Stay, and Crouch. With the development of game engines, it is becoming easier to implement them. Therefore, although the method presented in the above paper differs from the popular method, it is expected that there will be no great difficulty in applying it to the game because transplantation is easy. In the future, we plan to study to minimize the delay of each connection of the character motion so that the game can be optimized to best.

Keywords : Unity, Motion, Character, platformer, speed, response

Major Classification Code: Artificial Intelligence, etc

1. Introduction

Game development is the process of using graphics libraries, game engines, and image editing software to display images on the screen and let the player manipulate the images on the screen to create experiences and tell the player a story. In the early 80s, scrolling was a difficult task, and the developer had to face limitations such as CPU, memory capacity, and segmentation. Even with those challenges, some great side-scrollers were released, gracefully overcoming these limitations. However, in many cases, the motion was understandably simplistic or low-resolution It's remarkable that back in 1980, it used the default position-locking mechanism, keeping the car in focus at all times and the camera motion completely predictable (Keren, 2015). Even with th
ose challenges, some great side-scrollers were released, gracefully overcoming these limitations. However, in many cases, the motion was understandably simplistic or low-resolution (Bhosale, Kulkarni, Patankar, 2018).

* This Work was Supported by the Research Grant of the KODISA Scholarship Foundation in 2021

1 First Author. Dept. of Medical IT & Marketing, Eulji University. Korea, Email: d555v@naver.com
2 Corresponding Author. Director, LG Uplus corporation, Korea, Email: nevergiveup@lguplus.co.kr
3 Corresponding Author. Professor, Department of Medical IT, Eulji University, Korea, Email: lk04@eulji.ac.kr

Since then, character motion has been one of the main factors in the game. Game companies have their algorithms and scripts to prevent errors, but several Bugs and errors have a profound adverse effect on sales for game companies. In this paper, we introduce motion animators according to player input by using a simple game made by Unity engines. These are widely used by many companies, such as indies and large game companies.

2. Literature Review

2.1. Motion Response and Movement

Motion response plays a very important role in games. A good rule of thumb is that the game should always respond to the player's expectations. Characters that move according to various situations add more vitality when they are played. A lot of games have different motions and movements such as incrementally increasing, decreasing player velocity over time until they reach max speed. The following two logics are the most popular methods for motion response.

Title: Accelerating and decelerating pseudo-code

```
-- Get left/right input
    if left keyboard is Down then
      player.xVelocity = player.xVelocity + (-
      player.maxSpeed*player.acceleration*dt)  #Player go
      left with left velocity
    else if right keyboard is Down then
          player.xVelocity = player.xVelocity +
          (player.maxSpeed*player.acceleration*dt)  # Player
          go right with right velocity
    else
          if player.xVelocity < 0
          then
            player.xVelocity = player.xVelocity +
            (player.deceleration*player.maxSpeed*dt)
              if player.xVelocity > 0
            then
              player.xVelocity = 0
              #Player stop
                    end
          else if player.xVelocity > 0
            then
            player.xVelocity = player.xVelocity -
            (player.deceleration*player.maxSpeed*dt)
              if player.xVelocity < 0
            then
              player.xVelocity = 0
              #Player stop
                    end
           end
    end
```

Figure 1: Accelerating and decelerating pseudo-code

Figure 2 shows accelerating and decelerating methods. It is useful for making character motions and movements.

Acceleration is often divided into horizontal and vertical acceleration. If the character has horizontal acceleration, it takes some time for the character to reach the maximum running velocity, so it can be sluggish (Minkkinen, 2016).

Another popular method is the DeltaTime method. It is based on FPS (Frame per second) which is the interval of seconds at which physics and other fixed frame rate updates. It took to complete the last frame (jlett, 2011).

Title: DeltaTime method pseudo-code

```
Player.x = move character in X axis
If horizontal arrow key is pushed
   Player.x* player.basespeed * Time.deltaTime
   Weight = body.velocity
    movement = Player.x*body.velocity.y
   #Player move horizontal with it's frame rate
   end
```

Figure 2: DeltaTime method pseudo-code

Figure 3 shows a short but effective method. But it depends on the game's frame rate so, under 20 FPS, the motion will be stuttering (Hocking, 2018).

3. Related Research

3.1. Unity Game Engine

Unity, introduced in 2005, is a 3D/2D game engine and powerful cross-platform IDE for developers. The functions that are supported by Unity3D are very abundant. Unity3D produces applications based on JavaScript and/or C#. These are used to assign the animation or real-time transition of the Game-Objects defined in the application. The GUI of Unity3D helps a new developer to approach easily and script and program the transition of the Game-Object (Nachammai, & Senthil-Ganesan, 2018). As the game engine, Unity is able to provide many of the most important built-in features that make a game work. That means things like physics, 3D rendering, and collision detection. From a developer's perspective, this means that there is no need to reinvent the wheel. Rather than starting a new project by creating a new physics engine from

scratch–calculating every last movement of each material, or the way light should bounce off of different surfaces (Sinicki, 2021).

3.2. Colider 2D

In Unity3D engines, there are frequently used functions for collisions, especially OnCollision and OnTrigger. The difference between OnCollision and OnTrigger is whether the character can pass through the collision, and OnCollision cannot pass, but OnTrigger can pass (Kim, 2018).

Collider 2D components define the shape of a 2D GameObject for the purposes of physical collisions in OnCollision. A Collider, which is invisible, need not be the exact same shape as the GameObject's Mesh. In fact, a rough approximation is often more efficient and indistinguishable in gameplay (Unity, 2021).

Box Collider 2D is a Collider that interacts with the 2D physics system. It is a rectangle in shape with a defined position, width and height in the local coordinate space of a Sprite. Note that the rectangle is axis-aligned, with its edges parallel to the X or Y axes of local space. When enable **Used by Composite**, other properties disappear from the Box Collider 2D component, because they are now controlled by the attached Composite Collider 2D. The properties that disappear from the Box Collider 2D are **Material, Is Trigger, Used By Effector,** and **Edge Radius** (Unity, 2021).

Physics 2D Raycaster raycasts against 2D objects in the scene. This allows messages to be sent to 2D physics objects that implement event interfaces. The Camera GameObject needs to be used and will be added to the GameObject if the Physics 3D Raycaster is not added to the Camera GameObject (Unity, 2021).

3.3. Animator

An Animator Controller is a Unity asset that controls the logic of an animated GameObject. Within the Animator Controllers, there are States and Sub-State Machines that are linked together via Transitions. States are the representations of Animation Clips in the Animator. Transitions direct the flow of animation from one State to another. An Animator Controller allows you to arrange and maintain a set of Animation Clips and associated Animation Transitions for a character or object. In most cases, it is normal to have multiple animations and switch between them when certain game conditions occur. For example, you could switch from a walk Animation Clip to a jump Animation Clip whenever the spacebar is pressed. However, even if you only have a single Animation Clip, you still need to place it into an Animator Controller to use it on a GameObject. The Animator Controller has references to the Animation clips used within it and manages the various Animation Clips and the Transitions between them using a State Machine, which could be thought of as a flow-chart of Animation Clips and Transitions, or a simple program written in a visual programming language within Unity (Unity, 2021).

4. Motion Logic Experiment

4.1. Character Design

Unity engine is known for various assets that can be used in any project or game. In this paper, character design as shown in Figure 1 was conducted using an asset called "Warped City Assets".

Figure 3: Character Model

4.2. Motion Logic Design

The important thing about game character motion is that the motions connect with various keys at each time. Because of that, the character's motions were classified into Idle, Run, Jump, Shoot, and Crouch. Designing this motion, Animator 2D and C# script provided by the Unity engine were used.

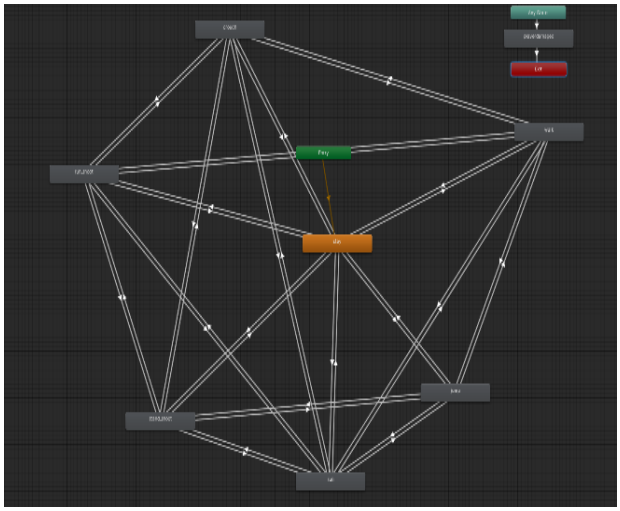

Figure 4: Animator Layer

In Figure 4, every arrow has bool type script that controls character movement. Each Bool type is designated as isWalking, isJumping, isRunning, isCrouch, isShooting. When the player presses the control key, the character moves according to the motion logic of Figure 5.

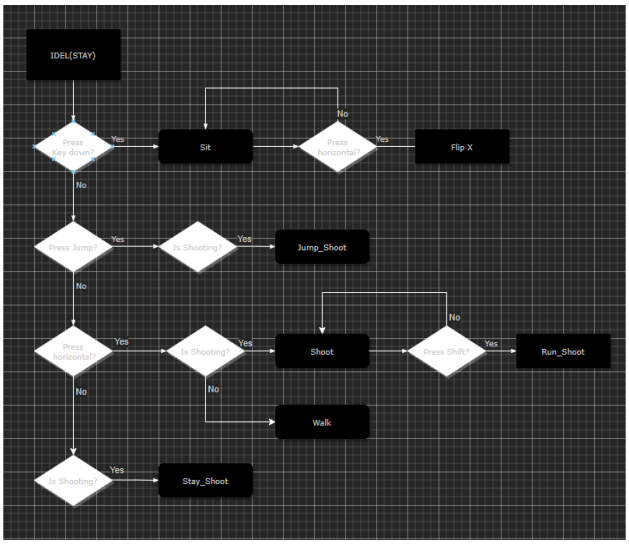

Figure 5: Motion Logic

BoxColider2D and PlatformColider2D are applied to the characters and platforms, and Raycast 2D is applied to the characters' collision decisions. When the distance between the character and the platform is 0.65f during the jump, the player is released into the Jump motion state. It comes over to the Stay state. The code to be implemented is as follows.

Title: Response of Speed method Pseudo-code

```
#Player
set Box2Dcolider (0.6x0.6)

#Walk
Player.speed = move character in X axis
if left arrow key is pushed
   Player.Xspeed = 1
if right arrow key is pushed
   Player.Xspeed = -1
if both arrow key is pushed
   Player.Xspeed = 0
   end
end

#Running
if shift key pushed
   Player.Xspeed = 2.5
   else if left arrow key is pushed
   Player.Xspeed = 2.5
   else if right arrow key is pushed
    Player.Xspeed = -2.5
   else if both arrow key is pushed
    Player.Xspeed = 0
   else
      player.speed = 1
end

#Crouch
If down arrow key is pushed
   Player.Xspeed = 0
   motion change in to crouch
   disable stand motion
   disable jump motion #unity animator bull control
   end

If down arrow key is not pushed
   motion change in to stand motion
   enable jump motion
   disable crouch motion
   #unity animator bull control
   end

#Jump
gravity = 1
If space key pushed
   Player.jumpPower = 3 #3 pixel
   Player.Yspeed = 2
   # go Y axis up and down in 2 second
   enable jump motion
   disable stand motion
   disable crouch motion
   #unity animator bull control
   end
   If platform distance < 0.65f
      disable jump motion
      enable stand motion
      enable crouch motion
    #unity animator bull control
   end
end
```

Figure 6: Response of Speed method Pseudo-code

4.3. Implementation

Figure 4 describes the basic logic in which the character reacts when entering a specific key on the keyboard. Entering the left turn-key based on the keyboard, the character speed is set to 1 and enables walking motion to the left. Entering the right turn-key, the character speed is set to -1 and enables walking motion to the right. When multiple keys are input at the same time, the speed is set to zero at the last motion and the same motion is repeated to prevent the character from taking a different motion from the last input key. When crouching, every speed sets to 0 and motion stays in a crouch position.

4.4. Test

The test compares with the response of the speed method introduced in the paper and two popular methods introduced so far. The comparison method uses macros built into the keyboard to manipulate characters at the same time, same speed, and same frictional force to input various keys.

After entering the key, two flags are set up on both sides to measure the speed at which the character goes from the A flag to the B flag to determine the reaction speed of each method. Flag A to B distance is 100 pixels and speed is 1. In theory, speed 1 sets a distance of 100 pixels to a distance of 1 second. The test proceeds 10 times, and the time and character reaction speed are determined by the average value. In the case of key reaction speed, the keyboard, and the game are linked to measuring the speed until the character reacts. All tests were conducted at a resolution of 1280×720 with a fixed 60 FPS.

5. Results

As a result, in the case of the Accelerating and Decelerating method, it took 1.11 seconds from flag A to B. When entering the keyboard, the character's time lag took 0.02 seconds. In the case of the Delta Time method, the time from A to B took 1.16 seconds and in the case of React Time, it took 0.03 seconds. The new method, Response of Speed, took 1.08 seconds and the reaction rate was 0.02 seconds.

Table 1: Testing react and speed about Method

Method	Flag A to B Time	Key react speed
Accelerating and decelerating	1.11 second	0.02 second
Delta Time	1.16 second	0.03 second
Response of Speed	1.08 second	0.02 second

6. Conclusion

This paper presents one of several ways in which the character shows natural motion without errors even if the character's control key is duplicated using the Unity engine. In the flag A to B time test, the Response of Speed method is 0.08 seconds faster than the Delta Time method and 0.03 seconds faster than the Accelerating and Decelerating method. In react speed test, the RoS method was the same time as the Accelerating and Decelerating method but faster than Delta time by 0.01 seconds. It concludes that the RoS method is better than the other 2 methods.

Since all games have their characteristics, it is difficult to be sure that they use the same motion logic presented in this paper. However, with the popularity of scrollers, many 2D cross-scroll games follow the formula of Dash, Shoot, Walk, Stay, and Crouch. With the development of game engines, it is becoming easier to implement them. Therefore, although the method presented in the above paper differs from the popular method, it is expected that there will be no great difficulty in applying it to the game because transplantation is easy. In the future, we plan to study to minimize the delay of each connection of the character motion so that the game can be optimized to best.

References

Bhosale, T., Kulkarni, S., Patankar N. S. (2018). 2D PLATFORMER GAME IN UNITY ENGINE. *International Research Journal of Engineering and Technology. 05(*04), 3021-3024. Retrieved from https://www.irjet.net/archives/V5/i4/IRJET-V5I4667.pdf

Hocking, J. (2018). Unity in Action. chapter-6/1 https://livebook.manning.com/book/unity-in-action-second- edition/chapter-6/1

Itay Keren (2015). *Scroll Back: The Theory and Practice of Cameras in Side-Scrollers.* Retrieved from https://docs.google.com/document/d/1iNSQIyNpVGHeak6isbP6AHdHD50gs8MNXF1GCf08efg/pub

Jellt (2011). *Advanced Platformer Movement*. Retrieved from https://www.instructables.com/Advanced-Platformer- Movement/

Kim, H. (2018). *Design of Runner Game using Overlap Circle in Unity3D.* Retrieved from https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART002377531

Minkkinen, T. (2016). *Basics of Platform Games*. Retrieved from https://core.ac.uk/download/pdf/80991297.pdf

Nachammai, P. M. L., Senthil-Ganesan T. M. (2018). 3D Game Development Using Unity Game Engine. *International Journal of Scientific & Engineering Research, 9(*3), 1353-1356

Sinicki, A. (2021). *What is Unity? Everything you need to know and roidauthority.* Retrieved from https://www.androidauthority.com/what-is-unity-1131558/

Unity Document 2021a*). Collider 2D*. Retrieved from https://docs.unity3d.com/Manual/Collider2D.html

Unity Document (2021b). *Box Collider 2D*. Retrieved from https://docs.unity3d.com/Manual/class-BoxCollider2D.html

Unity Document (2021c). *Physics 2D Raycaster*. Retrieved from https://docs.unity3d.com/Packages/com.unity.ug-ui@.0/manual/script-Physics2DRaycaster.html

Unity Document (2021d). Animator Controller. Retrieved from https://docs.unity3d.com/Manual/class-AnimatorController.html

Unity Learn (2021). Animator Controllers" Retrieved from https://learn.unity.com/tutorial/animator-controllers#5cdd9db3edbc2a1cdd86bfe5