ECS 279 : Final Project Report on Character Controller

Zixin Chi, Yu-Si Hsu, Sanjat Mishra, Pouneh B. Nikkhah

March 20, 2020

1 Abstract

Character controllers are crucial features in most games to allow players to interact with avatars that are in the game. In general, controllers take users input, control the movements and play animations. In this project, we implement the controller in third person view under Unity platform. We design a walking controller on a plane where the figure can move along an arbitrary path with walking, jumping, and attacking based on user's manipulation. We define the states transition and make sure that our third person character can react more seamless and smoother. In addition, we design an environment with lots of obstacles including walls and stairs, so that our third person character controller can interact with the environment and see whether it will have a permissive performance when encountering with walls or stairs.

2 Introduction

We implement a player-driven and third-person character controller and rigid body in 3D mode. The character controller can seamlessly perform different movements such as walking, jumping, and attacking in real time. The rigid body controller interactively reacts with diverse environments from user manipulation in real time.

Our characters are robustness since they can be demonstrated with moving to different directions including forwards, backwards, turning, etc., and unexpected slops and steps. Simulating a physic-based and biped locomotion is hard since bipeds are unstable and ensuring the characters move smoothly in real time is harder. Therefore, our project not only demonstrates the state transition of locomotion and movements but also specifically arms to find a solution to improve the smoothness and fluency of states transition that makes our third-person character perform seamlessly without delay and jerky movements. To improve the movement smoothness, we develop different states including idle, walking, punching and jumping state, and also provide the state transition which has the common frame of movements so that the character performs seamlessly with state and action changes. Furthermore, we create an environment with obstacles such as a wall and a stair and have our third-person character to interact with the environment.

In addition, we investigate the difference between character controller and rigid body controller for acting most situation of movements and show that the rigid body performs precisely with the environment containing obstacle and stair, comparing with the character controller. Character controller and rigid body both perform well in most of kinematic situations.

3 Related Work

Simulation of skilled movement of a character is a challenging problem, analytically and computationally. It involves modelling gaits, motions and reactions to different types of terrain. Several attempts have been made in the literature to develop realistic and natural human walking using mechanical models such as Kinematic Models, Physics-based Models, Reinforcement Learning, and Motion Imitation. Here we focus on the most closely related work in animation and Physics-based Models. Locomotion in particular has been the subject of considerable work, with robust controllers being developed for both human and nonhuman characters, e.g., [1], [2], [3]. Many such controllers are the products of an underlying simplified model and an optimization process, where a compact set of parameters are tuned in order to achieve the desired behaviors [4], [5], [6]. Dynamics-aware optimization methods based on quadratic programming have also been applied to develop locomotion controllers [7][8]. While model-based methods have been shown to be effective for a variety of skills, they tend to struggle with more dynamics motions that require long-term planning, as well as contact-rich motions. Trajectory optimization has been explored for synthesizing physically-plausible motions for a variety of tasks and characters [9], [10]. These methods synthesize motions over an extended time horizon using an offline optimization process, where the equations of motion are enforced as constraints. Recent work has extended such techniques into online model-predictive control methods [11], [12], although they remain limited in both motion quality and capacity for long-term planning. The principal advantage of our method over the above approaches is that of generality. We demonstrate that a single model-free framework is capable of a wider range of motion skills (from walks to highly dynamic kicks and flips) and an ability to sequence these; the ability to combine motion imitation and task-related demands; compact and fast-to-compute control policies; and the ability to leverage rich high-dimensional state and environment descriptions. Simple Biped Locomotion Control, or SIMBICON, is one of the frameworks that attempts to solve these challenges. SIMBICON can generate various types of gaits and movement, including hopping, jumping and running. Additionally, SIMBICON uses feedback error learning from motion capture data to achieve motions [2].The downside of this technique is that the whole process of controller generation is not fully automated, which means it requires experienced tuning to make the gait look natural, and it doesn't include the reaction time delay into the model. Another popular technique is to use preexisting animated controller from Mixamo and improve based on that. For example, since we decide to use character controller instead of rigid body in our project, we can introduce multiple controllers to interact with each other and create obstacles for the controllers to climb and jump.

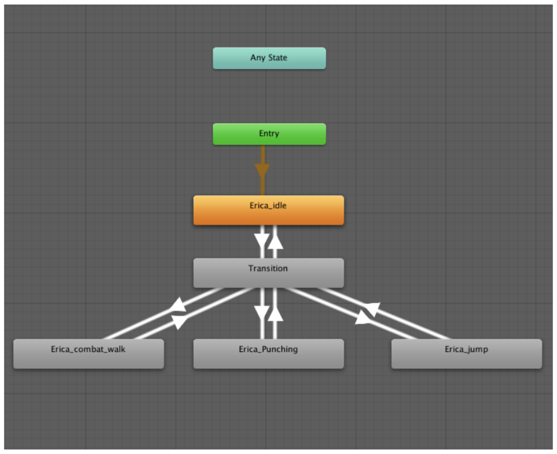
4 Technical approach

The idea is to create a character controller for every character on the scene and design interactions with various different objects like walls, obstacles or other characters. The main advantage of choosing Character Controllers over RigidBody is that character controllers leave room for more player-driven interactions which gives the player more explicit control over the character.

At each point in the game, the character is said to be in a state i.e, it performs a specific action at every give time. In this project, we model several different states of the character based on its interaction with in-game objects. Moreover, we propose seamless state transitions in order to provide a smoother game play and fluid movement of the character throughout the scene. In order to do so, we plan on providing state transitions only for states which have common frames during action. This allows for a seamless change in state and action.

Below are some of the states we plan to implement for the character object:

1. Idle State
2. Walking State
3. punching State
4. Jumping State



The Idle state is the state that the character is in by default when the game starts or when the character is not performing any other action.The walking State is when the "W" key is pressed down, the character moves into the Walking state. A condition variable is set to 1 which denotes the transition to the walking state. A boolean variable called "running" is set to true when the character enters "Walking" state and set to false otherwise.

Idle(left) and walking(right) state

Punching State is when the Left mouse button is clicked, the character moves into the Punching state. A condition variable is set to 2 which denotes the transition to the punching state. A boolean variable called "punching" is set to true when the character enters "Punching" state and set to false otherwise.Jumping State is when the right mouse button is clicked, the character moves into the Jumping state. A condition variable is set to 3 which denotes the transition to the Jumping state. A boolean variable called "Jumping" is set to true when the character enters "Jumping" state and set to false otherwise.

Punch(left) and jump(right) state

we are also interested in evaluating the difference between the character controller and rigid body. we tried to add some vertical stairs and obstacles so that it can be tested with different scenarios such as how it collides with objects, the jumping mechanic and falling mechanics.

The character controller is a component that can be added in Unity. It's used to allow the character move around according to the environment. In our project, we modeled the character controller as a upright capsule shaped object for simplicity. There is a square box attached to the object so when we can tell which direction it is facing when it rotates. The movement is computed based on the slides methods. One of the advantages we noticed is that a capsule shaped object smooths jump up/down actions when it's over an mesh obstacle or some stairs(shown below). In addition to the basic moving function, we add gravity factor so that when the character controller jumps and falls from a stair, it looks more naturally.

The build-in nav mesh obstacle in Unity allows collision detection control so that each contact and return the contact positions, contact normal vectors and penetration depths. Thus, each contact will suggest that the capsule moved into a wall and reached the limit.

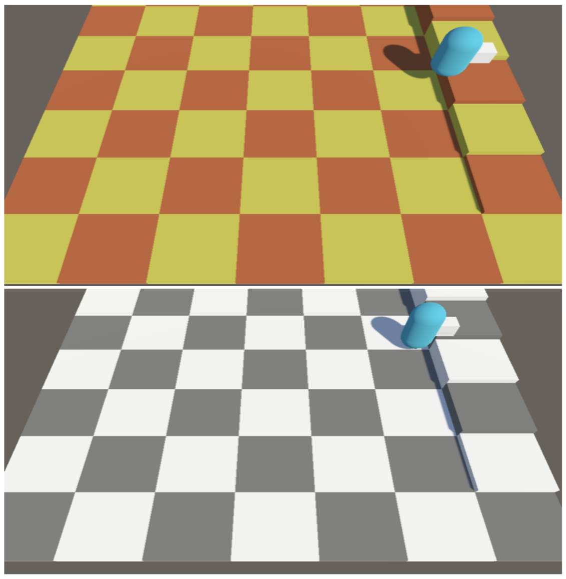
we also implemented dash and drag function so that the controller can jump forward or backward based on user's input. The method is to scale the forwarding direction vector with a dash vector. The dash vector is dependent on the dash distance and the drag. A drag force is used to decelerate the controller, just like friction force in the real world. And then we apply the velocity vector =/ 1 + Drag force*delta time to get the real velocity on each direction.

Then we implemented a rigid body under the same scene. So instead of a character controller component, we create a capsule collide object and a rigid body which locks the x and z axis. However, instead of the traditional update function, we use a new function to realize the physics update. To achieve a more convincing simulation, the calculation is done more smoothly by restraining the frame rate too low or too high. To apply a force on a rigid body, we want to use instant reaction regardless of the mass of the object. And when moving or jumping based on user's input, there is acceleration time to reach to target speed.

5 Experiments and Discussion

We focus on evaluating our model on motion switching, and motion smoothness. For motion switching, we want to make sure the character reacts correctly with the environment and switches to different motions between walking, running, standing jump as well as running jump. Furthermore, we smooth the motion transition between different controllers to interact with each other. Motion switching measurement can be computed between initial state with goal state or location at interactive rates. We include velocities in the continuous motion states to ensure smoothness and handle motions with velocity-dependent actions such as running. In addition, we will include obstacles for characters to perform climbing and jumping.

For the character controller and rigid body collider, there are some differences in how they handle the reaction with the surroundings. The rigid body will react very precisely and use the physics material property to calculate the reaction, while the character controller is more tolerable. For example the character controller will sometimes climb the slops and stairs without jumping.


Character Controller v.s. Rigid body

As shown above unlike character controller, rigid body is not able to climb up the obstacle stair. We think this is very useful when designing game characters and maps. If one wants to design a real-world simulation map where everything is as real as possible, a rigid body may be better to detect small physical obstacles and produce animation based on this. The rigid body provides more functions in Unity to interact with the physics. We also ask some people to use our character controller to gain feedback and improve smoothness. In the first few trials we set the speed and jumping gravity too high so that it doesn't feel natural enough.Also we change the ground color to a grid pattern to provide more contrast of the movement.

6 Technology Stack

The following technologies are going to be employed in the proposed project:

1. Unity 2019.3.0f6
2. Visual Studio 2019 for C Scripting

7 Timeline and Contribution

- Week 1 Set up the basics of the project and define a rigid-body and the Idle state(Yu-Si Hsu)
- Week 2 Implement basic movement, i.e walking and running on a rigid body(Pouneh Bahrami)
- Week 3 Design side movement, detecting obstacles(Zixin Chi Sanjat Mishra)
- Week 4 Implement "Jumping". and improve overall smoothness (Zixin Chi Sanjat Mishra)
- Week 5 Evaluate the system and investigate its bugs and finish report (all members)

8 Future work

During the implementation phase of this project, we came up more ideas that can be realized in the future. For example, give the character controller the ability to push an object. When the controller stands on an object, its weight can be used to push the supporting object downward. To push horizontally, the controller applies an impulse to the object. The pushing speed is dependent on the drag force of the pushing object. In addition, we can design a moving platform such as elevator or a conveyor so that the character controller can move on the platform. In order to do so, the character needs to check the ground location and compute the absolute velocity with respect to all ground plane. For the complex animation character, such as the knight we found from MAXIMO, we plan to add transition animations so that it looks more natural between different states.

9 Conclusion

In this project, first we implement a character controller that allows a user to control the motion of a character such as walking, jumping and attacking. It's able to walk along an arbitrary path. Next, we investigate on the different between the character controller and the rigid body. We find that character controller is more permissive when facing an obstacle such as a wall or stairs, whereas the rigid body can react very precisely and use the physics property to calculate the reaction. However it requires a lot of effort for rigid body to do fine tuning work. Both can achieve same animation in most cases, so it is crucial to think about the main focus before picking one of them.

References

[1] Stelian Coros, Philippe Beaudoin, and Michiel Van de Panne. Generalized biped walking control. *ACM Transactions on Graphics (TOG)*, 29(4):1–9, 2010.

[2] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.

[3] Yuting Ye and C Karen Liu. Synthesis of responsive motion using a dynamic model. In *Computer Graphics Forum*, volume 29, pages 555–562. Wiley Online Library, 2010.

[4] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. Diverse motion variations for physics-based character animation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 37–44, 2013.

[5] Qing Zhang, Son Tung Ha, Xinfeng Liu, Tze Chien Sum, and Qihua Xiong. Room-temperature near-infrared high-q perovskite whispering-gallery planar nanolasers. *Nano letters*, 14(10):5995–6001, 2014.

[6] Meng Wang, G Audi, AH Wapstra, FG Kondev, M MacCormick, X Xu, and B Pfeiffer. The ame2012 atomic mass evaluation. *Chinese Physics C*, 36(12):1603, 2012.

[7] Gary P Scavone, Antoine Lefebvre, and Andrey R da Silva. Measurement of vocal-tract influence during saxophone performance. *The Journal of the Acoustical Society of America*, 123(4):2391–2400, 2008.

[8] Yoonsang Lee, Moon Seok Park, Taesoo Kwon, and Jehee Lee. Locomotion control for many-muscle humanoids. *ACM Transactions on Graphics (TOG)*, 33(6):1–11, 2014.

[9] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

[10] Kevin Wampler, Zoran Popović, and Jovan Popović. Generalizing locomotion style to new animals with inverse optimal regression. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014.

[11] Perttu Hämäläinen, Joose Rajamäki, and C Karen Liu. Online control of simulated humanoids using particle belief propagation. *ACM Transactions on Graphics (TOG)*, 34(4):1–13, 2015.

[12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.