h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Faculty of Media

Bachelor of Arts, Animation and Game
Second Semester

Project: **DeadCare**

Logline: **Prevent various children from killing themselves before and during their play in a cozy 3D isometric daycare room with a lot of potential for chaos.**

Lukas Salewsky, 1117683, 05.07.2023

Introduction

With our project “DeadCare” we contribute to this semesters project-topic “One Screen, one Set”, in which we set ourselves the goal of making a game with no camera-switches and no to complicated story while it should provide fun playing. We decided to work on the idea of having a room which we as the player can visit from the outside, while there are children, doing more or less reasonable actions. We as the player must watch out for certain dangerous actions that the children might perform. There are various ways we can do that, either by taping a dangerous object, distract the child with cookies or manually stop the wardrobe from falling over.

This project had many disciplines involved, both on the academic, as also on the practical side. To first cover the academic relevance, there was an uncountable amount of research to be done. With some little exceptions like the next destination a child decides to go to, there are no coincidences, which means that every part of this game had to be well thought out, not to destroy the experience. Therefore, especially when for example we were adding new mechanics to the game or exported assets from another software, we always had to inform ourselves about something completely new, and we had to do it in a clean way to best possible avoid technical issues.

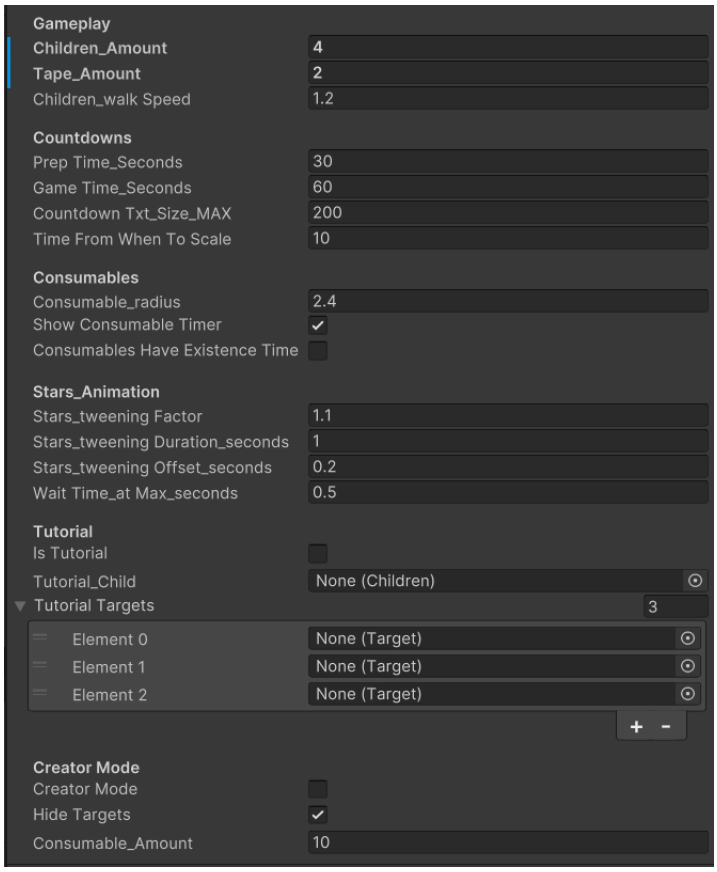
Of course, practical relevance is also given. We potentially could implement all of the theoretical learnings from the study-course into our game, and for topics like setting up the NavMesh for children-AI and for encapsulating scripts that worked out very well. Also, we already must learn how to deal with working in a team. Even though being the only programmer might be an unrealistic scenario I could learn a lot about the importance of dividing tasks based on skills and interests.

Individual Contributions

A+G Design:

The design part took around a tenth of my work for the project. At some point of progress, we were in the situation of deciding about the first UI-visual elements as an upgrade to the already existing buttons which held text in them. So, I came up with some basic ideas for how the UI-Elements like the tape-roll or the consumables should look like. Even though my drawings were not chosen as the final ones, the basic shape behind them remained. A bigger role in the design-contribution was my approach to set up a working in-Unity sandbox-tool for both the level-designer as well as the animator. It should be as easy as possible for them to work with our project. For the level designer, Anastasia, I made up some rules about how to scale the floor (incl. NavMesh-rebake), how to implement obstacles like the wardrobe, how to connect that obstacle to a NavMesh-target, so that the agent knows at its destination, which obstacle it stands in front of and many more. For the animator, Zoe, I wrote a script, which handles all animations from one place. So it was not that complicated anymore to combined two animations into one (which was necessary, because the Children-animation and the Obstacle-Animation are two separate), bending their playtime dependent on the individually set target-time, but also it was much easier to set up single animations like the children tape-Removal or the eating-Animation.

Furthermore, I set up a setting-script, from which all kinds of Game-Design-relevant decisions could be managed, like the children-speed, if we are currently in debug-mode, how long a game-phase should be, and many more. There also is a tutorial function implemented that, if toggled on lets a specific child walk to the destination the Game-Designer wishes, instead of choosing a random destination. That saved a lot of time when we had to debug.



Settings_Script: tweaking possibilities in the Unity inspector

Game Development and/or Technical Art:

Game Development, or programming was my main part in the project, therefore I'll give 80% of my time to it. My focus on this project was script encapsulation. That means, while every script should have its purpose in the program, they should be as loosely coupled as possible. There are two reasons. For one, which I recognized early, it is very easy to find functions you want to change or add new mechanics. For example, the RayCast-Script does only recognize the current object which is clicked and the current three-dimensional-mouse position, while it communicates for example with the Object_moveHandler-Script to move an obstacle in the room. It also communicates with the Animation-Scripts attached to the clicked objects to tell them for example to rewind the animation using a play speed of -1.
The menuHandler-Script just serves the purpose of managing data between the gameplay-rounds, while this one communicates with the canvas-Script to update what the player can see. Here I make use of the Model-View-Controller.

Another big part of programming took the simulation of the NavMesh agents. While Unity provides the user with a well-functioning NavMesh-Navigation-Tool[3], getting multiple agents in a room to walk randomized to a currently opened target was still a big challenge, because the agents are constantly moving around, having different wait-times at different targets and therefore might be blocked for some seconds before they can move to the next open target.

Also, agents should avoid going to the same destination. For me it worked out pretty well with the solution of having two states for each possible target, which are the Booleans isOpen and isTargeted. IsOpen tells an agent if there is a child on the target currently. IsTargeted means that a child currently is going to this target. If IsOpen is false, or IsTargeted is true, the other agents will ignore that specific target. The agents always recalculate their path as soon as they have done their task (e.g., waiting at a wait target). If after that no target is currently reachable, the agents fall into an isStopped-Mode, which frequently checks for new open targets. Meanwhile the agent holds its position at the current target and keeps isOpen true. That's the reason, why there is both an isOpen and isTargeted Boolean.

Another complicated task was to implement the consumable-mechanic. If you throw in a plate of cookies, the child should immediately cancel the current action and go there (at least, when it is walking to a destination or it is in a danger-zone, but not, when it is at a non-deadly zone). There were several things to do for getting the cancellation right, like resetting the current timer, if there was one at that moment, changing the animation, resetting the old animation from the obstacle, letting the children recognize it stands in the consumable if the player decides to throw it right into the child instead of just putting it next to the child so then it is in the radius-zone, and lots more.

A+G Methodology/Producing and Production Management:

This section got 10 percent of my work. Before the synthesis paper task was cancelled, I was very curious about different pathfinding methods, because we also wanted to have one of them in our game to have the children move through the room. So, I read through different scientific papers which provided information not only about differences between the algorithms but also explained why the A-Star-Pathfinding turned out to be the most used one, and how this algorithm specifically works. Also, multi-agent handling and its difficulties were discussed[1, 2]. I found out that Unity provides a handy method to implement the A-Star-Algorithm in combination with a baked NavMesh by typing keywords like pathfinding and Unity into an internet search-engine.[3] Through this way of research I often directly was offered a link to the official Unity documentation[4], in which I found lots of insights into the functionality behind the build in software tools, not just for NavMesh with the agents and obstacles, but also for RayCasting[5], especially how my code should look like, when I want to translate a 3D world space mouse click into a 2D screen-position and vice versa.
Also, I was able to take a deeper look into how the RigidBody component[6] works and why the inbuild physics-system might cause trouble when not set up correctly.
Here the Unity forum also came in handy. They have answers to any questions regarding working in unity. Sometimes, especially when physics are involved, unforeseen behavior occurs. To find out through internet research, that you might face a common problem, and you find out why that happens can be very rewarding (in this case there was a RigidBody-setup mistake).

Analysis and Conclusion

Bugs were caused, because of Variable state changes without having an option to reset them in other areas of the code where the state should be reset. This can be caused by any area of code for any game. E.g. the state of an animation that should be currently play might be set active, but if an old state still is playing its old animation, it might continue with the old one. That's especially relevant when the old animation is looped. This can cause the character to stay stuck in the old animation loop, but not play the new one, which is waiting for the old one to end. Another example would be NavMesh-Logic. Because in a closed NavMesh-System the agents are dependent on existing open targets they can walk to. If one agent targets a specific destination, that target should be closed for all other agents. That in mind, if the state of the target does not change to open again if the agent leaves the target for a new one, then the old target will be closed for ever. Therefore it is not reachable again for any other agent, which can quickly cause all agents to stick to their old position. Just re-opening an old target if the agents have a new destination is not enough. There could also be an event which causes the agent to get away from the target, and there are more examples, which lead to the explained issues.

What makes this a big problem is that some of these special causes are not found immediately after implementation. Later, when playtesting the game, bugs might occur.
I often had to search for these kinds of errors in code, which took lots of time.
Because of this and many other examples it would be better next time to write concrete documentation about which states have which dependencies and when a new gameplay-technique is built in the code, a better plan made beforehand will probably safe lots of time trying to solve bugs later on.

I knew that anything would be possible to implement with time, so in our planning phase I was very open to all the creative ideas about what would fit into our game.
That's why we collected lots of ideas and had in the end a clear vision of the project, how it would work and what makes it a game. Even though we were able to implement nearly all our ideas in the end, there were some unforeseen difficulties involved. As I already knew beforehand, I would be the bottleneck in our production, because the implementation of animations, the level-design, UI, having options to tweak gameplay and lots more, must be made possible first. In the beginning I started working on the core-gameplay-mechanics. Parallel the other team members started producing ideas on paper and digital assets and animations.
While I was very concentrated on getting the core game run, the amount of finished creative work grew. Suddenly, when everything seemed to work, we tried to implement the first assets and animations. While the assets don't really interact with the game, the animations have caused big issues because of export-settings. Trying to solve that problem at this step is way too late, because besides the animations, lots of other stuff should be included as well at this time. So I had to close the bottleneck again and work on that one thing.
This problem was complicated to solve, because we already thought, we were far in production when suddenly a very new problem arrived and we had to debug it in an environment we did not want to change at that state. Next time I work in a team as a programmer I rather first concentrate on the way everything concerning creativity should be implemented instead of working on the actual game. For example the implementation of the animations would have been much easier, if I had made up a specific scene in Unity for the animator to test their newly made animation before I start with the gameplay-mechanics. So if the animator has his first animation done he can already setup his implementation in Unity and work on occurring problems, while I can keep on working on the game.

Conclusion and Future Work

I have learned that imports into the game engine should happen as soon as there is material available to test it with. And then create an early workflow for that, which is written down to some sort of instruction/documentation. Then it will not be much of an act to replicate those steps which have been already successfully worked before. I've also learned that whenever I have a break at the weekend, and even if it's just one day off work, it could improve the whole of next week's productivity. When you can lean back for some time and reflect on the project it gives new inspirations and broadens the possibilities. For my future work, now especially knowledge about the different famous programming patterns should be gained quickly to see, in which cases I could have written better code, or could have had a clearer architecture. I already recognized by now that certain implementations in code could have been done more efficient, for example the Animation-Script could maybe somehow also have worked with Unity's async[7] which would have checked if the animation finished and then call something.

List of References

1. Cui, Xiao. and Shi, Hao. "A*-based Pathfinding in Modern Computer Games." Researchgate. January. 2011. https://www.researchgate.net/profile/Xiao-Cui-12/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games/links/54fd73740cf270426d125adc/A-based-Pathfinding-in-Modern-Computer-Games.pdf
2. Lawande, Sharmad Rajnish. Jasmine. Anbarasi and Izhar. "A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games." Mdpi, 28 May. 2022. https://www.mdpi.com/2076-3417/12/11/5499
3. Unity Technologies. "Building a NavMesh." Unity. 2023. https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html
4. Unity Technologies. "Unity Documentation." Unity. 2023. https://docs.unity.com/
5. Unity Technologies. "Physics.RayCast." Unity. 2023. https://docs.unity3d.com/ScriptReference/Physics.Raycast.html
6. Unity Technologies. "Rigidbody." Unity. 2023. https://docs.unity3d.com/ScriptReference/Rigidbody.html
7. Tarkdev. "Unity async / await: Coroutine's Hot Sister [C# & Unity]." YouTube. https://www.youtube.com/watch?v=WY-mk-ZGAq8&list=PLKeKudbESdcx2vFx-aEpu0UmZFelV5C8pK&index=2

Appendices

List of relevant items:

- Animation-Script
- Settings-Script
- RayCast-Script
- Object_MoveHandler-Script
- MenuHandler-Script
- Canvas-Script

List of third-party elements used in the project:

- DOTween (Unity sprite animation asset)
  https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676