Game Development on Unity

Berkay Torun
Istanbul Kultur University, Department of Computer Engineering, Istanbul/Turkey
berkayk3211@gmail.com

Serdar Karakurt
Istanbul Kultur University, Department of Computer Engineering, Istanbul/Turkey
serdarkarakurt6@gmail.com

Taha Bilal Aydın
Istanbul Kultur University, Department of Computer Engineering, Istanbul/Turkey
tahaaydn@gmail.com

Yusuf Altunel
LC Waikiki, Istanbul Kultur University, Department of Computer Engineering, Istanbul/Turkey
yusuf.altunel@LCWaikiki.com

Abstract
In this paper we present the result of our work on implementation of an interactive environment that can be used to develop adaptive games with animating characters. The environment is designed to run in design and run mode, whereas certain design mode characteristics are still applicable in run mode so that the player can enhance the game and keep the features for future playing sessions. In design mode, the designer, can create a whole new world with animating characters, define rules using a predefined syntax to set certain features, restrictions and conditions. The designer can run the game in simulation mode, visualize the results, make corrections, package and deploy the game when satisfied with no need to write even a single line of code.

The approach presented here provides much more flexibility comparing with static games fixed at compile time. Interactive development environment with combination of modifiability gives the player the power to let the game be evolved in time without waiting the game developers to deploy a new version. Additionally, this flexibility reduces the stress on the game vendors arises from dealing with high and varying expectations of players within a limited time and development resources. Finally, the environment can be used to design special purpose games for teaching, healing, or any other gamification case study.
Keywords: Game development

Introduction

The game players would like to be able to make small enhancements in the games they play. As the games are fixed at the time of deployment, it is quite difficult to collect and add such enhancement requests to the games. Usually, this can only be done with a new release which takes time and resources. By adding the ability to make such enhancements to the running games, the whole process of small enhancement request is simplified and optimized. Additionally, adding dynamically maintainable rules to the game, makes the game development more convenient and flexible. This can increase the speed of special purpose game development and reduce the coding effort enhancing the game design process. We named this game “Rule the World (RTW)”.

The aim of the study is to develop an interactive environment to develop games with animation characters. The designer will be able to define new animation characters, set certain characteristics, restrictions and conditions using a rule definition syntax. The rules will be interpreted and executed on the fly. So, the designer can set the animation to perform according to the rules, run in simulation mode and visualize the possible enhancement options. These rules are stored in a rule-base to let the user be able to make changes as necessary.

The aim is to implement an interactive environment to design adaptive game environment, define rules to interact with game objects, visualize the effect running in simulation mode and make corrections. Basically, two types of execution options are realized, the first one is design mode and the second one play mode. In design mode certain abilities are possible including add or remove components, define a ruleset for users to interact with the game objects and characters. The designer can run the game in a simulation mode after setting up the rules. As the user modifies the rules, the system parses and interprets them on the fly and change the behavior of the game, so it is possible to realize the effect of such changes. This is quite easier and more straightforward for game designers in developing games comparing with the coding effort required in traditional approaches,

compilation and deployment needs, which in return requires additional programming competencies or specialized work force.

The developed system can be used to add new objects, change the environment, let the rules be independent from the source code and change them to be effective during the simulation of play mode and keep the changes for next play sessions. These features help for faster development game development, reducing the coding effort and let the special purpose games be easily created and evolved for future needs. So, teaching with gamification, curing diseases setting up exercises with playing games and similar needs can be easily satisfied.

Related Work

This section contains information about other projects and articles that has been useful during the development of RTW. However, we haven't found a game with high similarity with our game. Instead we looked for simulation type of games and basic game development guides.

Computer games have always evolved toward increased technical complexity to give the players things they have never experienced before (Blow, 2004). RTW provides players a simulation where they create their own playing style. Each wave of games is attempting several technical feats that are mysterious and unproven (Blow, 2004). RTW is also an attempt to create a new playing style for simulation type of games. Thus, brings a new structure, new functions and a new approach for how to play the game. With this approach, a lot of new issues appeared. Gameplay planning, the ability of both designer and players and how to manage them.

We had a few similarities in concept, so we took game Rimworld as a reference which is a simulation game that is based on colony management (Sharma & Pal, 2015). In Rimworld you gather characters with different skills that affect their job and manage them in a way, so the colony gets bigger and bigger while defending against various scenarios. The things we inspired from this game are basic things such as their UI but also major things like individual character AI and its usage. In Rimworld, when a character gathers resources they carry it to a stockpile that is designated by the player and these gathered resources appear only if they are in a stockpile. We were inspired by this simple UI feature and used in our project. Rimworld characters mostly move automatically according to their job but the player can also assign them a one-time task so the character will prioritize it and finish it first before doing its normal job.

We wanted to have autonomous character movement in project for better aspect of a simulation game thus we used A* search algorithm in our character movement and pathfinding algorithm. A* algorithms is a graph search algorithm for finding path from characters initial position to given or searched object's position (Sharma & Pal, 2015). A* algorithm prioritizes nodes close to goal and closest to starting point. If there is path exist between start and goal A* guaranteed to find that path with effectively. g(n) represents the exact cost from starting point to any point n, h(n) represents the estimated cost from point n to the destination.

f(n)=g(n)+h(n) is the formulation of cost of general path to goal for A* algorithm (Xiao & Hao, 2011).

Using Unity to develop a game for inexperienced developers is the most practical solution (Brett& Simons, 2017). Game engine itself is easy to understand and learn and there are a lot of projects going on with the engine. Which is community being there to find solutions and improve overall quality of engine. There are phases in game development which involves many skills and knowledge required to do so. Unity goes through these phases by supplying various components and features such as collision detections which require mathematical calculations on screen and images. Unity does this in the background and gives developers a better and easier interface to work with.

There are many different games in our age and time. Some of these games are developed on custom game engine and frameworks while someone them are is being developed with private game engines of big game companies. Either way accessing these game engines are easier said than done. Therefore, the Unity as a free game engine is one of the most known and used game engine and framework in the game development field. Our project's graphic rendering and code execution are all done in the Unity.

The Study

This section explains the details of our game project, Rule the World.

Interpreter is a program to translate high-level language programs into low-level language that can be executed by computers statement by statement as each instruction is executed (Xiao & Xu, 2011). In this manner, the interpreter we designed for our problem is to translate human language to high-level programming language.

Interpreter implemented for this project, it is required to identify English writing rules and high-level programming language writing rules such as functional separator implemented as 'comma' character and functional parameters divided by 'white space' character. Their work is like our implementation, but differences are they implemented their program to understand C language symbols and it aims for High-level programming language to Low-level programming language translation. It is designed as it should be as close as possible to speaking language thus it will allow the player to give commands to character easily. There are many concepts taken consideration while implementing a custom object oriented interpreted language such as Advanced Generics Bounding (bind parameters by their methods or fields as well as their types), Properties (plug-gable reserved words), Meta (advanced reflection with system listeners), Default values for Method Parameters, Custom Lambda declaration (naming a combination of parameters and return type for ease of use), Methods declared as Objects (also declared as Lambdas) (Singh & Agrawal, 2018). In our project we implemented the part of Advanced generics bounding by separating functions with 'comma' character and separating function parameters by 'white space' character. We implemented our interpreter with C# language, and it does not have standalone features and only works as in game functionality.

The game separates users to two, Player and Designer. Designer is one the who puts objects like houses, trees or even characters in the world. Designer also writes rules to decide what Player can and cannot. Designer enters the basic operation(s) in a text box as a string. These operation(s) have equivalent functions to them. These operation(s) are divided with comma while function parameters divided with white space characters. Designer can define repetition giving a number before colon.

5 times : Find AppleTree , GatherObjects Apple and Log

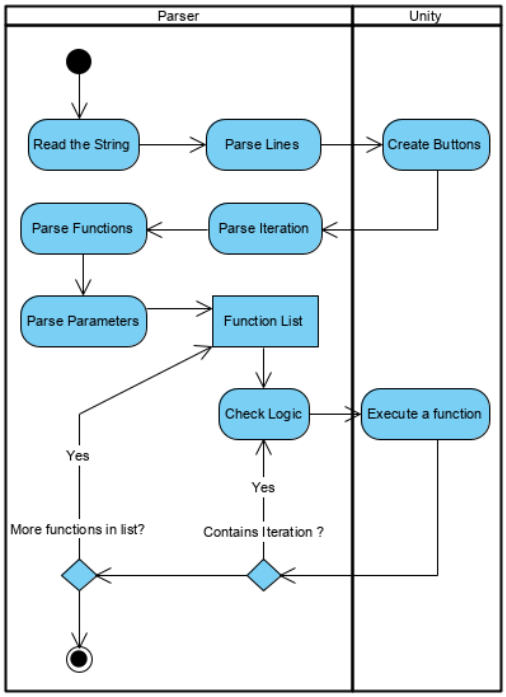

Figure 1: *Example function to find apple tree and gather apples.*

Activity diagram at [Figure 1] shows how this string will turn in to the rules that Player will be using. The divided function will be called and processed. Functions after parse operation saved on a list <string> are called one by one after completion with or without parameters depending on the parsing operation. Function may have a waiting condition such as walking until reaching the destination thus syntax should work accordingly. Each function finished in the list either stops the program or the user can input the repetition of the same rule until a new rule is set.

Regular Expressions library from C# is used for picking an expression from a text for parser and this way, the system works with string inputs from the user to command the characters. Commanding mainly works with a parser system, rule inputs are taken from the player as string. Application parse this string into methods and their parameters. Then starts to action in order. If needed, one waits for another to finish.

We included Aron Granberg's pathfinding library which is a A* pathfinding algorithm navigation mesh

generator to grant pathfinding to our characters in the game (Sharma & Pal, 2015), (Granberg, 2020), (Rimworld, 2020). With this library we generate mesh graphs for each new object created or removed from the game asynchronously and with efficient performance. Characters then move in the world according to this mesh, checking if there is an object in its path or is there even a possible path to the position. Another library package we included was TileMapExtras. With TileMapExtras you can make an automatic tile selection script which is used to select tiles according to their surrounding tiles. Meaning that you can say that while single water tile is surrounded by other tiles, other tiles must follow a certain rule we set in script and blend in with water so it would not look out of place. This makes world creation much easier.

In Unity, there is built-in methods for every script. They are named Start, Awake and Update. Start and Awake methods are used for initializing and setting certain objects and components on game objects, Start runs when object is created while Awake is executed when game is first executed. Update method is used for checking or updating variables on each frame game runs through. These methods are used throughout the project. For class system, we worked around a GameManager that controls the game flow, settings, and many other operations. Every character and object that are included in game and objects added to the game by users and ButtonHandler which is used for handling buttons are controlled by GameManager class.

Items and resources are kept in an Item Database System. Characters have their own inventories that are shown in the interface and those inventories are connected to Item Database. Time based actions like interaction timers, respawn timers etc. are handled with a Time Tick System. Generic insertion of object is defined as text files inside resources folder with specific texture files included. These text files are read by program to create button for designer to add these objects in world. Gameplay properties and configurations are set through to json file inside project folder with the name “config” text files. This file is read as a json object from program and set properties of built in objects and time scale of game. Game world is two dimensional and we need to give feeling of depth to players thus, we used the y value of position of game objects to define which game object is in front of each other. This is called as sorting order of objects. In example if y value of character is lower than y value of a tree game will show character in front of tree otherwise game will show character behind tree.

We implemented a configuration text file to save and restore general configuration of game such as tick count per second and built in object properties. This is done by creating a new class inside unity and setting it member variables as configuration variables. These class will be converted to json object and written to a text file inside project folder. If such configuration file exists in project files on next loading of game state it will be read from file and proper configuration object will be created and its properties will be used in game objects.

Objects can be placed and removed in designer mode. To make this work we check mouse position on each frame if it is on a deployable tile or not. This function will run on each frame and if function cannot be finished within frame it will be skipped. Game will be run at average 30 so there will be enough time to finish this function with in a second. We hold information about the object is going to be placed inside game manager. When designer clicks on the button it will send object to game manager then tile will check if mouse over deployable tile and game object successfully send to game manager and object is created and deployed on mouse position by using left mouse button.

We used unity’s editor to have brush feature on world creation. This feature can only be used on development state so we create object which designer add from text files are created inside game world outside of player view on deactivate state. Each time designer places object it will be copied from this hidden object and its components will be set and configured.

Findings

In implementation, the objects can be imported from external files, so the game world can be extended by such objects infinitively. It is possible to define how to interact with these objects. Additionally, the objects can be stored as characters in the inventory to make them active and let them behave and animate. The interactions and behavior of characters are defined using syntactically specified rules. Such rules can be quite complex depending on the need of the designer and player. A remarkable feature of our approach is that the rules can be parsed and executed on the fly to change the game in the run or simulation mode.

The toy project implemented in this study shows us that, the game development environments can be designed to provide more flexibility to the designers and players. This helps the designers to reflect their creativity to the games with no or limited coding effort. For the gamer point of view, this means shorter time of receiving new features to the games. Additionally, letting the rules be modified during the play mode and ability to save them for next time, the players themselves get the chance to make such enhancement to the game and even share them

to other players under copyright restrictions of the game vendors. These features can bring totally a new play development and evolution industry based on sharing and collective game development. Reducing the effort of code and making game development environment more efficient and flexible, can help specifically designed games, such as teaching by gamification and game based medical treatment to have more possibilities.

Conclusions

In this study the aim was to show that a flexible game development environment can be implemented to create player's own world with dynamically manageable rules. We used "Rule the World (RTW)" environment as an example to show our approach to create adaptive and evolvable games are realizable.

The initial implementation is just a Proof of Work project with limited functions, simple rule syntax and minimum management options. Currently, characters in game has less than 10 functions but they can be combined with each other to create new rules, however, the possible number of functions can be radically increased to make it possible to define rich number of rules. In future, it is possible to other game types adding different objects and more complex and realistic rules. For the RTW environment, additional features such as day/night mode, user-tile map, chopping timers and animations can easily be added. However, additional to RTW, other game types can be implemented. The environment can be used in professional training options, education and any other situation that can be enhanced by adapting gamification.

References

Sharma, K. S. & Pal, B.L. (2015). Shortest Path Searching for Road Network using A* Algorithm (pp. 513-522). International Journal of Computer Science and Mobile Computing, Vol.4 Issue.7.

Blow, J. (2004). Game Development: Harder Than You Think: Ten or twenty years ago it was all fun and games. Now it's blood, sweat, and code (pp. 28–37). Queue 1, 10. DOI:https://doi.org/10.1145/971564.971590

Xiao, C. & Hao, S. (2011). A*-based Pathfinding in Modern Computer Games, (pp. 125-130) IJCSNS International Journal of Computer Science and Network Security, Vol.11 No.1.

Brett, J. & Simons, A. (2017). Implementation of the Unity Engine for Developing 2D Mobile Games in Consideration of Start-Up/Student Developers (pp 271-278). International Conference on Technologies for E-Learning and Digital Entertainment.

Xiao, X. & Xu, Y. (2011). The Design and Implementation of C-like Language Interpreter (pp.104-107). International Symposium on Intelligence Information Processing and Trusted Computing.

Singh, A. & Agrawal, A. P. (2018). LIPI: A Custom Object-Oriented Interpreted Programming Language (pp.538-543). 8th International Conference on Cloud Computing, Data Science & Engineering.

Granberg, A. (2020). A* Pathfinding Project, https://arongranberg.com/astar/ : Accessed (30/05/2020 16:33).

Rimworld (2020). RimWorld - sci-fi colony sim , https://rimworldgame.com : Accessed (28/5/2020 14.07).