# **Graybox Game Development in Unity**

Content

1. Introduction
2. Design
3. Implementation

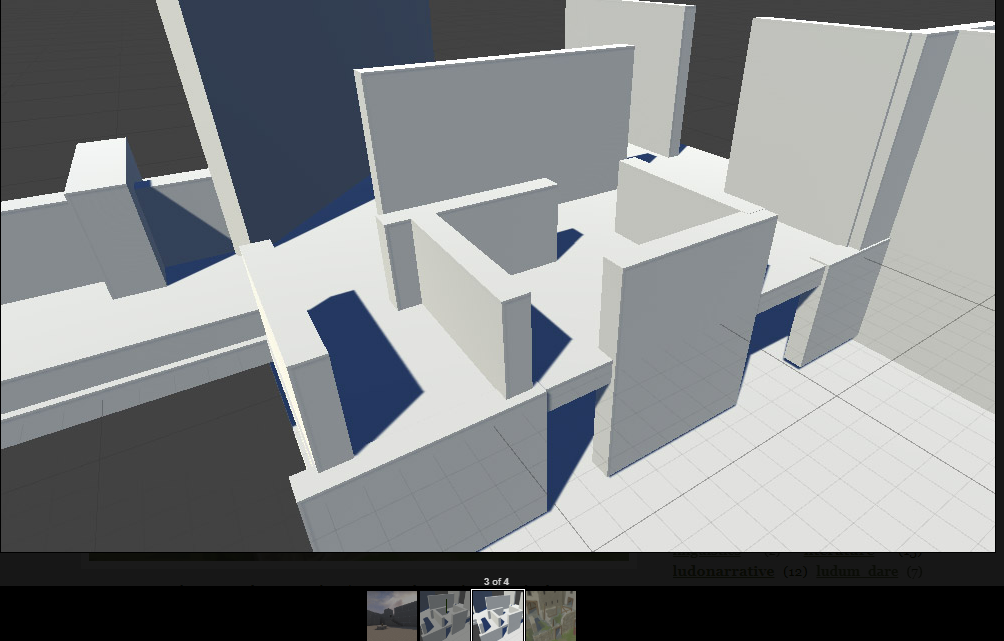
4.References

Abstract

Gameplay programming is vital for video game development and benefits from good tools and techniques. However, techniques are still used in the industry that involves describing how the computer operates. Reactive programming is a way to describe eventful and stateful computer programs declaratively, focusing on what the program should accomplish. This thesis uses the reactive library UniRx with the game engine Unity 5 to create an FPS with reactive techniques, and discusses the advantages and disadvantages of these. Object-oriented reactive programming is used to combine reactive objects with Unity’s component-based framework without using wrappers for non-reactive objects. The results include static methods for observable user input, patterns when defining game components with reactive content, and communication between game objects through interface components. It can be concluded that reactive programming for game programming enables the programmer to describe and understand game logic in a declarative manner

1.INTRDUCTION

Grayboxing is a level design practice where you build a rough block-out version of your level using blocks (usually gray boxes) so that you can iterate and test the layout as soon as possible. Almost every 3D game engine has some sort of box primitive tool -- if you know how to use that, then you can graybox.  
  
Before you graybox, you must make sure you've established a general game design direction. You should generally know how this level might fit into your game or workflow. There's no point in grayboxing if you don't even know what the player should be doing, or what this level is supposed to convey. Is the level supposed to be easy or hard? Does it focus on combat or non-combat? Should it feel scary or safe? Level design must always exist in the context of a larger game design, or else you're just wasting your time.  
  
Then, open up your 3D game engine, and let's start laying down some boxes…



For this example, I'll be using Unity, but there's no reason you can't do a similar thing in Unreal or any other 3D toolkit. Whatever you use, try to choose an actual game engine, and not solely a 3D art tool like Maya or SketchUp, because you'll want to be able to walk around in the actual space in-game.

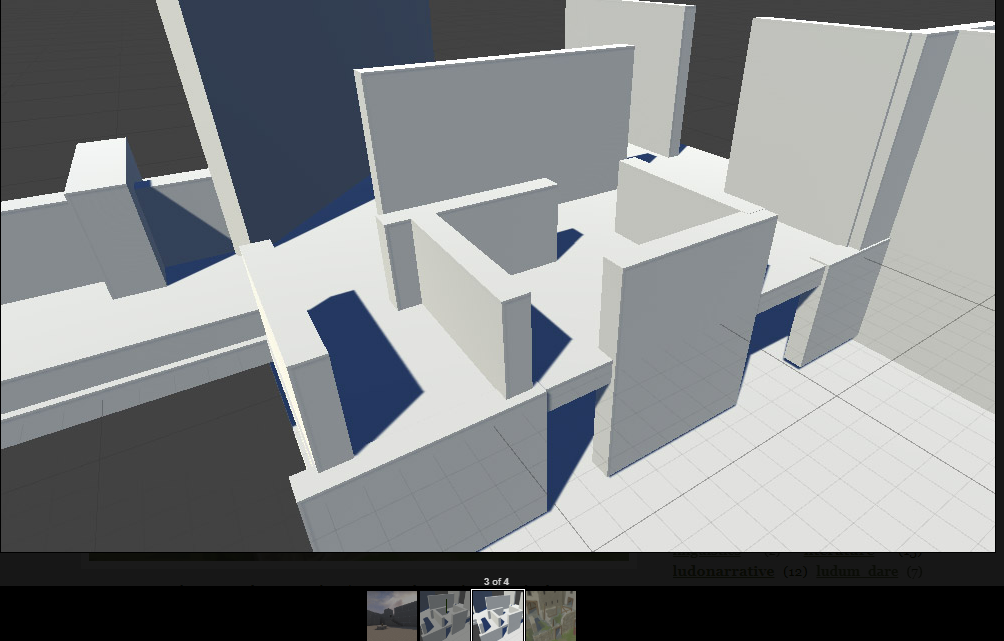
1. **It helps to sketch something.** Like, even a 1 minute scribble on a napkin will help you plan your level. You don't have to follow the plan, it's just to help your ideas flow better.
2. **Add some kind of basic floor** plane to your scene. In Unity, I like placing a wide but thin cube at (0, 0, 0). It helps me get started with laying down some geometry.
3. **Add a scale reference**. You'll want something that's roughly "humanoid sized" to help you figure out how big to make walls / how wide to make hallways / etc. Ideally, you use some sort of actual 3D character, but in Unity I often use simple capsules resting on the floor. By default, these capsules are 2 units tall and 1 unit wide.
4. **Add more blocks.** When you have one wall placed down, duplicate that wall object and rotate it to make another wall, very quickly. Keep cloning objects and rearranging them until you have a room. Try to stay fast and loose, and spend only a few minutes on this. Unless you're working with a BSP-based engine like a Quake or Source Engine, or with a specific modular kit like for Skyrim or Fallout 4, then don't worry too much about aligning things to a grid.
5. **Test as soon as possible!**Add a player controller object so you can walk around, and test as soon as you have a room ready. If you have NPCs, powerups, items, etc. then place some of those too. As you play, ask yourself whether the spaces feel "right" and whether it will support the gameplay you want. (e.g. is there enough cover for a cover shooter, is there enough strafing space for an arcade shooter?)
6. **Based on the results of your playtest, keep adding to your level and iterate.**Don't forget to add more humanoid scale references in your new area, and make sure you regularly fly around the 3D viewport and keep checking different angles / line of sight. Adjust and fiddle as necessary, but also don't fixate too much on one area.

Observable user input

Unity’s methods for accessing user input is adapted to be observable in a reactive manner. UniRx does not provide that in its current version. As Unity’s input methods were static, thus globally accessible throughout the program, the observable input is made static as well. The observable input was placed in static classes where each class was categorised by the type of input device it managed, like keyboard, mouse, and joystick. This was to make it clear and apparent where different types of user input were accessed. In Unity 5, the standard way to acquire input from keyboard, mouse, and joystick is through each game loop. The state of input devices can be checked for each update by using methods from the static class Input in Unity’s API. This was utilized with the static method Observable.EveryUpdate(), provided by UniRx, to make the input as observables. The operation DistinctUntilChanged() was used to get input values only at change, preventing the generation of redundant input values at every update. The operation Skip(1) was used to skip the first emitted value from all inputs. If not performed, a value is issued when the program starts without the detection of any user input. The static methods for different input actions are designed to include observable values as arguments. A method that returns key presses from a specific key would be based on the observable key that is passed to the method. If the key name later changes, the observable value that was previously returned, is changed to emit key presses from the new key.

Design Images

1.Layou



1. Skeleton View



References:

1- Sirbu, D. (2003). Digital Exploration of Inbuilt Architecture: A Non-Photorealistic Approach. ACADIA

2003, pp. 235 - 245.

2- Stappers, P. J., Saakes, D., & Adriaanse, J. (2001). On the Narrative Structure of Virtual Reality: An Analysis and Proposed Design. CAAD Futures 2001, pp. 125 - 138.

3- Kruijff, E., & Donath, D. (2000). Supporting Shared Architectural Understanding: Spatial knowledge transfer within Virtual Environments. Proceedings of ACCOLADE 2000 Workshop.

4- Dorta, Tomás ; Augmented Sketches and Models: The Hybrid Ideation Space as a Cognitive Artifact for Conceptual Design , University of Montreal, Faculty of Environmental Design.

5-Young,R. Michael , O.Riedl,Mark, Branly,Mark , Jhala,Arnav , Martin R.J. , Saretto C.J. ; An architecture for integrating plan-based behavior generation with interactive game environments ; Submitted for Publication in Journal of Game Development.

6- Abdullah Bakarman , Ahmed ; Quality Evaluation Tool for the Design Studio Practice , King Saud University, Riyadh, Saudi Arabia.

7-Lowe,Russell , Newton, Sidney ; Using an Analytics Engine to Understand the Design and Construction of Domestic Buildings ; Faculty of Built Environment, The University of New South Wales, Sydney, 2052,Australia.

8- E. Haque, Mohammed , Dasgupta, Pallab ; Architectural/Engineering Visualization using Game Engine ; Proceedings of the 2008 ASEE Gulf-Southwest Annual Conference , The University of New Mexico – Albuquerque.

9- Conway, Kevin R. ; Game Engines for Architectural Visualization in Design , University of Washington

2011, P9-13.

10- Paul Gee, James ;Video Games, Mind, and Learning , University of Wisconsin-Madison , 2003, P2.

11- Koehler, Tanja, Dieckmann, Andreas ,Russell, Peter ; An Evaluation of Contemporary Game Engines , RWTH Aachen University, Germany , P743-749.

12- Campbell, Dace A. ; A Critique of Virtual Reality in the Architectural Design Process , University of Washington ,Seattle, WA 91895.

13- Miliano ,Vito, Industrielle, Perilith ; Unreality: Application of a 3D Game Engine to Enhance the Design, Visualization and Presentation of Commercial Real Estate.

14- Moloney, Jules , Harvey, Lawrence ; Visualizatio a d Au alizatio of A hite tu al Desig i a Ga e Engine Based Collaborative Virtual Environment , University of Auckland and the Spatial Information Architecture Laboratory, RMIT University.