УДК 004.514

DESIGNING ABILITY SYSTEM FOR A REAL-TIME ACTION GAME

Horiushko S.D.
Scientific supervisor – senior lecturer Novikov Y.S.
Kharkiv National University of Radio Electronics
14 Nauky ave., Kharkiv, Ukraine, 61166, Department of Software Engineering,
Tel. +38 (057) 702-14-46, e-mail: stanislav.horiushko@nure.ua

Action games with a combat system often contain abilities for the purpose of increasing gameplay variety and/or as a part of character progression. This task might prove challenging, as there are a lot of requirements to such a system, since it is an essential part of gameplay and should be able to accommodate a wide variety of abilities, added at the stage of initial development as well as in the content updates which might come further in the game's lifecycle.

Taking into consideration the general demands for an ability system, we can formulate the core requirements:

1. The system should be able to accommodate abilities with vastly different logic and effects.
2. The system should be open to extension should the need for new types of abilities arise in the future.
3. The abilities should be uniform and interchangeable.

Within the scope of this work, we will add a number of additional functional and technical requirements for the design of the ability system:

1. Abilities should have a “cast time” – a delay after which they are used, during which no other actions can be performed, and a “drawback time” – a similar delay after using the ability.
2. Abilities should have a “cooldown” – a period of time for which the same ability cannot be cast again.
3. Following subtypes of abilities should be implemented:
   a. Player ability – contains the ability name, description, in-game icon and cost, can only be cast when cost requirements are met.
   b. Channelling ability – can be “channelled” for a period of time, continuously applying its effect.
   c. Ability chain – uses pre-made abilities to be cast in a sequence.
4. Target platform – Unity 3D game engine using C#.

Since the logic behind different abilities may vary greatly, it is a good decision to implement each ability in its own class, which will contain its functionality. Instances of this class will be created and assigned to the corresponding character when an ability is made available to them. To make abilities uniform and minimize code duplication, the abilities should inherit a common class, which contains their shared functionality and serves as an interface, through which other entities can access them.

As the logic behind abilities is implemented inside individual classes, core and subtype functionality should be extracted into abstract classes [1], from which all the other inherit. The resulting structure is described in a partial class diagram (see Figure 1). The individual abilities directly inherit from any of the abstract classes, depending on the subtype they belong to.

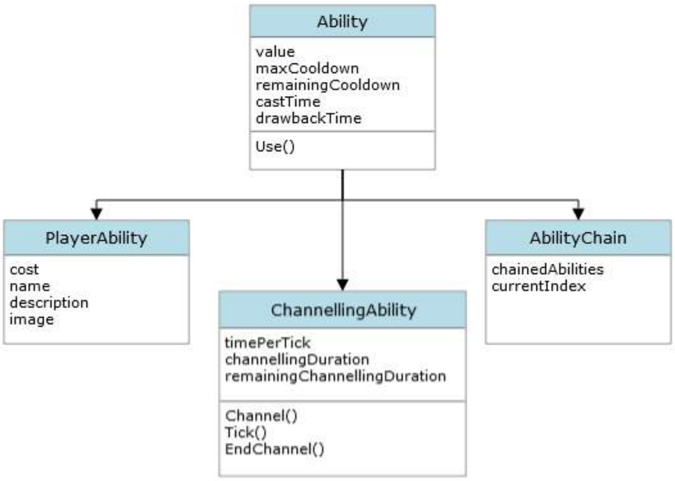

Figure 1 – Partial class diagram for abstract classes

The next step is designing a system, which adds delay before and after using an ability, restricting character action. To implement this, we need to store the ability the character will use until the cast time runs out, and then use it. While the character is inside this timeframe, his actions are restricted. The resulting algorithm is implemented inside the Character class (see Figure 2).

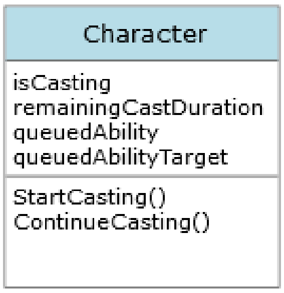

Figure 2 – Character class

The ContinueCasting() method call is repeated every iteration of the game engine cycle until the cast is finished.

References:

1. abstract (C# Reference): https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract (date of access: 12.04.2023).