Доклад за проект Make 15

Георги Хърлев

ООП C# 2021/2022

Съдържание:

Избор на дизайн

Тъй като проектът представлява игра, то водещата мисъл в дизайна беше “Какво бих искал да видя в интерфейс на игра, която искам да играя”. От тук дойде идеята за бутоните, които са достъпни от менюто на играта – Find Match, който имплементира онлайн игра между двама играчи; PVE, който имплементира офлайн игра срещу компютър и Help бутон, който ще даде начални насоки на потребителя как да използва интерфейса.

Преди игрите реших, че би било интересно играчът да може да избира сам името си (ако ли не – то ще му бъде предложено случайно генерирано име от тип “Player”+ (Число от [1,99])).

Воден от горната мисъл, дизайнът на самата игра между двама играчи се състои от чат между тях, който показва съобщението и от кого е пратено, като има цветова индикация за името на пратилия (синьо за пратените от играча, червено от опонента, зелено от сървъра). Добавена е комуникация от страна на сървъра за кога играчът е свързан към играта, кога е загубена връзката с опонента и кога играта приключва (кога има победител или равенство). В левия горен ъгъл са показани имената на двамата играчи. Има 3 видими списъка с цифри – това са на текущия играч (централно най-отгоре), на опонента (под нея) и голям списък за наличните избори от цифри. Избирането става лесно, като първо се маркира желаното число от играча на ход и се натиска бутон “Pick”. За да няма нужда от помнене чий рунд е в момента е добавена и червена стрелка, която сочи към полето на играча, чийто рунд е сега.

Цветовата гама на играта като цяло беше решена на принцип какво би било изглеждало най-интересно – не много натрапващо и изморително за гледане, но не и скучно безцветно. Така реших гамата да е около жълто-оранжево и леки цветове като синьо-бяло за по-фоновите елементи.

Имплементация

Като избор за имплементация на сървър имаше немалко варианти. В моя се имплементира Duplex договор между сървъра и клиента, чрез който, освен че клиентът може да праща заявки, сървърът също може да праща на клиента. Това беше нужно, за да се сигнализира за намерен опонент, за изгубена връзка, за синхронизиране на списъците от числа и предаване на съобщения.

Режимът на паралелност е *Reentrant ,* тъй като използвам само една нишка за обработка на двамата клиенти (повече за това в четвърта точка на доклада) и се използва само една инстанция за всички клиенти, което позволява достъпването на данни да става по-лесно, тъй като те са до голяма степен общи за двамата играчи. За пазене на данни реших отговорността да е на други класове – singleton Database, а за пазенето на данните на отделен играч – class Player. Спазено е правилото да се връщат копия на данните на референтните обекти (специално за листовете с числа), за да няма нерегламентиран достъп до тях извън сървъра.

Връзките към клиентите се пазят в Речник с ключ идентификатора на играча и стойност връзката.

Винаги при опит за извикване на функция от клиентите се проверява дали това е успешно, а при грешка – изтичане на времето за връзка или друго, тази връзка бива изтрита.

При връзка на нов играч се проверява състоянието на другите играчи - ако то не е добро, те се премахват. При наличие на двама активни играча се започва игра, а тези след тях не биват свързани (и биват уведомени за това). При излизане на играч от игра се рестартират данните и при наличие на нов играч, той се свързва с първия чакащ.

Ограничения на проекта и тестване

Най-голямото ограничение е възможността да се играе само от двама играча. Сегашният еднонишков дизайн не е подходящ за много игри, тъй като е възможно забавяне на връзката със сървъра, заради последователното изпълнение на заявките.

Проектът е тестван както за нормално стартиране и спиране на програмата, така и за неочаквано такова. При нормална работа, при затваряне на програмата, връзката със сървъра ще бъде изтрита, както и данните за играча. При неочаквано, като загуба на интернет връзка или неочаквано спиране (или crash) на програмата, сървърът ще получи грешка при опит за връзка с клиента и той ще бъде премахнат като неактивен. При липса на връзка или неработещ сървър, програмата ще даде съобщение за неуспешна връзка към сървъра и ще се върне към началното меню.

Срещнати проблеми по време на изработката

Един от проблемите беше, че първоначалната идея беше за многонишков сървър, работещ с много играчи (поддържащ много игри). Тази идея беше ограничена до игра само на двама. Редуващите се рундове позволиха последователно изпълнение на операциите, тъй като никоя от тях не е прекалено тежка да забави изпълнението. Единствен проблем може да е забавяне заради връзката към даден играч, но решението на този проблем няма да промени резултата – вероятно прекратяване на играта, заради проблем с един от играчите (или забавянето няма да зависи от работата на сървъра, тъй като изчакване ще бъде необходимо и в двата случая). Това доведе до идеята за *ConcurrencyMode.Single* ,което беше променено на *Reentrant* ,за да реши *deadlock* проблемите с едновременните заявки.

(Този проблем е решен по същия начин и за класа на клиена).

Другият проблем беше за пазене на данните в сървъра. Можех или да го направя със Singleton клас, или да използвам *InstanceContextMode.Single* . В решението си и двете са налице, тъй като бях имплементирал първата версия вече.