Wordel

Cilj ovog rada je bio napraviti repliku web aplikacije Wordle koju je napravio *The New York Times*.

C# već sadrži .NET Multi-platform App UI (nadalje MAUI), no s obzirom da je za razvojno okruženje bio korišten *Linux*, MAUI nije bio dobar odabir.

Za izradbu su korišteni C# programski jezik i Avalonia UI (nadalje AUI) platforma (engl. framework) za razvoj korisničkog sučelja. Avalonia UI podržava Windows, MacOS i Linux desktop operativne sustave, kao i iOS i Android, te uz njih i WebAssembly (web preglednike). Površinom pokrivenosti podržanih platformi je time sličnija JavaFX platformi koja je prethodno bila dio jezgrenog Java kompleta za razvoj softvera (engl. Java SDK), dok ju Oracle nije odlučio ukloniti kako bi smanjili veličinu JDKa povodom uvođenja modula.

Učitavanje resursa uključenih u projekt

Dok se za rad s datotekama izmjnjivog sadržaja može koristiti `System.IO.File`, u kontekstu razvoja softvera često želimo uključiti datoteke za koje ne očekujemo izmjene od strane krajnjih korisnika aplikacije (engl. end users).

S obzirom da aplikacija treba sadržavati popis riječi za provjeru korisničkog unosa, to je izvedeno čišćenjem i pretvorbom postojećeg rječnika (autor: Goran Igaly) u JSON format.

Za učitavanje JSON datoteke je korištena `Newtonsoft.Json` biblioteka u `Wordel.Model.Game.WordList`.

Zanimljivost s kojom sam se susreo tokom učitavanja resursa (datoteka uključenih u assembly projekta) je da se za njih ne bi trebala koristiti `Assembly#GetManifestResourceStream(String)` funkcija nego `IAssetLoader#Open(Uri)` funkcija koju pruža AUI.

`GetManifestResourceStream` funkcija bi trebala raditi na različitim platformama, no `IAssetLoader#Open` dozvoljava pohranu u predmemoriju (engl. caching) i automatsko skaliranje slikovnih resursa ovisno o DPIu uređaja koji pokreće aplikaciju (engl. DPI based texture scaling).

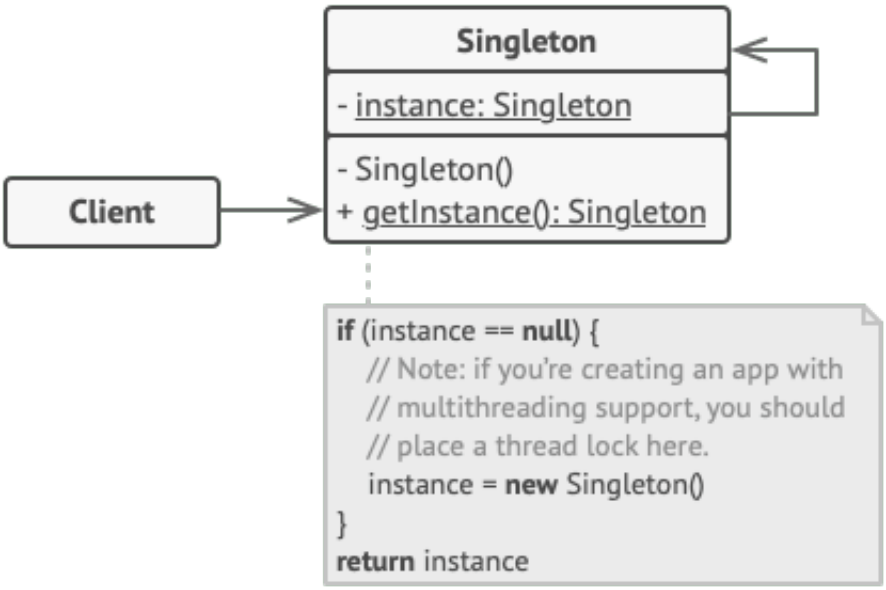
Uporaba `IAssetLoader#Open` u mojoj aplikaciji nije značajna, no za naprednije aplikacije koje rade s većim brojem tekstura i zbog drukčijih zahtjeva područja primjene, ona dozvoljava ubrzanje izvršavanja koda zbog smanjenja pristupa datotečnom sustavu.

Jedinstveno instancirana klasa

S obzirom da je rječnik statičan u kontekstu izvođenja aplikacije, koristio sam statičnu klasu (`Wordel.Model.Game.WordList`) sa statičnim članom tipa `string[]?` za pohranu liste riječi koji je inicijalno postavljen na `null` te mu se

nakon prvog pristupa učitavaju vrijenosti iz resursa `dictionary.json` priloženog uz aplikaciju.

U klasničnoj primjeni GoF "singleton" obrasca stvaranja bi klasa imala funkciju za pristup jedinstveno instanciranoj statičnoj instanci (ili objektu) klase, no s obzirom na jednokranu svrhu klase `WordList` (za pohranu `string[]`) sam odlučio učiniti sam član statičnim. Tako je dobivena funkcionalnost sličnija Rustovoj std::sync::Once strukturi. `static` tip klase uključuje da je `WordList` također i `sealed` (hrv. zapečaćena) klasa, tj. da nije dozvoljeno njeno daljnje proširivanje. C# za razliku od Kotlina (trenutno) nema `object` ključnu riječ za automatsko generiranje bajtkoda (engl. bytecode) koji provodi identičnu logiku opisanu GoF obrascem.



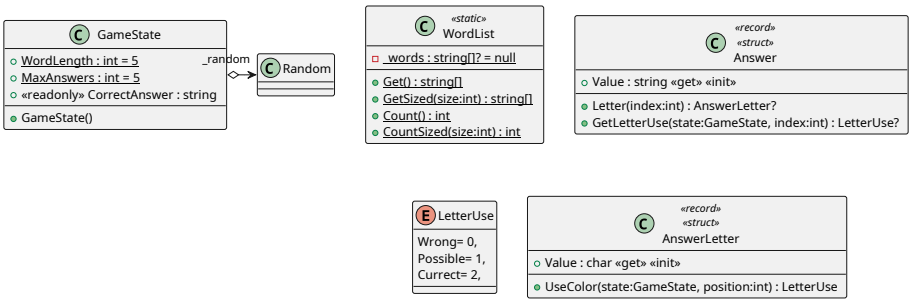
Slika 1: Struktura jedinstveno instancirane klase

Izrada dijagrama

Za izradu UML dijagrama je bio korišten PlantUmlClassDiagramGenerator (autor: Hirotada Kobayashi), PlantUML C# import i Umbrello nisu radili.

UML dijagram modela aplikacije

Zaključci



Slika 2: Dijagram aplikacije