Tower Defense - zápočtový program

Andrej Pečimúth

Programování v C++, ZS 2019/2020

1 Používateľská príručka

1.1 Inštalácia

Najprv potrebujeme zostaviť knižnicu SFML. Predpokladá sa operačný systém Windows 10 a Visual Studio 2019. Hra by ale mala byť zostaviteľná na každej platforme podporovanej knižnicou SFML.

1.1.1 Zostavenie SFML

1. Zložku SFML-2.5.x otvorte vo Visual Studiu.
2. Vo Visual Studiu kliknite na CMakeLists.txt.
3. Zvoľte konfiguráciu, napríklad x64-Debug, a stlačte F5.

1.1.2 Zostavenie TD

1. Otvorte súbor TD.sln vo Visual Studiu.
2. Otvorte Properties pre projekt TD.

   V sekcií C++/General editujte Additional Include Directiories, aby zahŕňal hlavičkové súbory SFML podľa vzoru nižšie. Prefix `C:\path\to\repo` nahraďte skutočnou cestou k SFML. Vzor:
   `C:\path\to\repo\SFML-2.5.x\include%(AdditionalIncludeDirectories)}`

   V sekcií Linker/General editujte položku Additional Library. Časť `x64-Debug` nahraďte požadovanou konfiguráciou, ktorú ste zvolili pri zostavení SFML. Vzor:
   `C:\path\to\repo\SFML-2.5.x\out\build\x64-Debug\lib;`
   `%(AdditionalLibraryDirectories)`

   V sekcií Debugging editujte položku Environment, aby mal program prístup k potrebným .dll súborom. Vzor:
   `PATH=\%PATH\%;C:\path\to\repo\SFML-2.5.x\out\build\x64-Debug\lib;`
   `C:\path\to\repo\SFML-2.5.x\extlibs\bin\x6\$(LocalDebuggerEnvironment)`
3. Zvoľte konfiguráciu a stlačte F5.

1.2 Úvod

V hre tower defense sa bránite vlnám prichádzajúcim protivníkom. Protivníci prichádzajú vo vlnách po vopred vyznačenej ceste. Každá vlna sa skladá z nejakého počtu robotov alebo dronov. Prichádzajú z ľavého okraja mapy a prechádzajú na pravý okraj. Vašou úlohou je stavať veže, ktoré same strielajú na prichádzajúcich protivníkov. Veže sa dajú kúpiť za peniaze, ktoré sa získavajú zneškodnením protivníkov. Keď sa protivník dostane za pravý okraj mapy, uberie niekoľko zásahových bodov. Dosiahnutím nekladného počtu bodov končíte. Veže je možné za polovicu ceny predať, prípadne vylepšiť. Každá vlna protivníkov je silnejšia alebo početnejšia ako predchádzajúca.

1.3 Ovládanie

Po spustení programu stlačte ľubovoľnú klávesu pre začiatok hry. Hra sa ovláda ľavým tlačítkom myši. Kliknutím na prázdne políčko (zelené) otvoríte ponuku veží. Kliknutím na ikonu veže si ju zakúpite (ak máte dostatok prostriedkov). Kliknutím na nejakú zakúpenú vežu sa zobrazia možnosti pre danú vežu. Na pravej strane tlačítko pre predaj, prípade tlačítko pre vylepšenie na ľavej strane.

1.4 Rozhranie

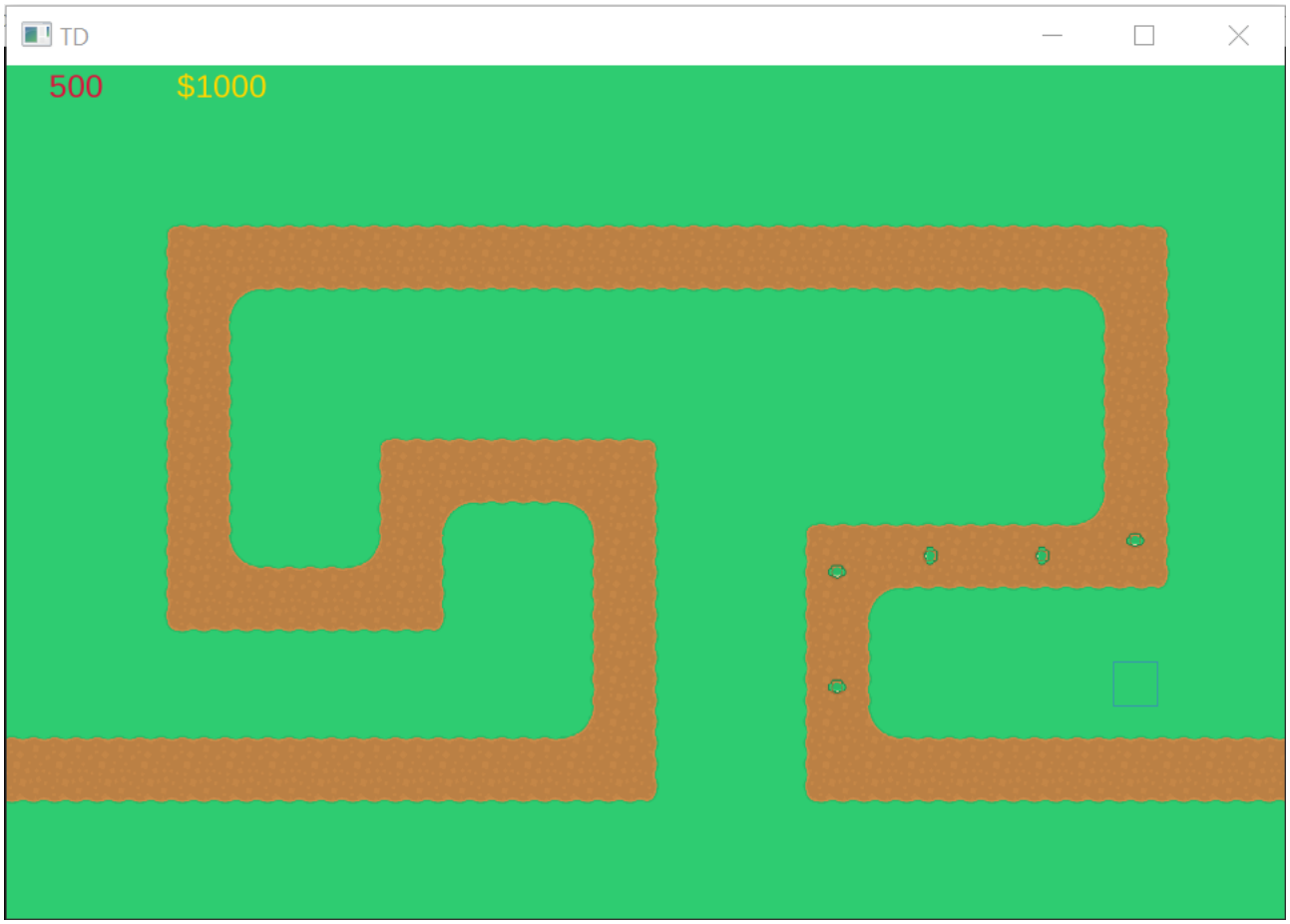

Figure 1: Mapa a používateľské rozhranie

Červené číslo v ľavom hornom rohu udáva počet aktuálnych zásahových bodov. Žlté číslo napravo od neho je aktuálne množstvo peňazí, za ktoré sa dajú kupovať veže. Hráč má na

začiatku 500 zásahových bodov a $1000. Protivníci prichádzajú zľava, pohybujú sa po hnedej ceste a vychádzajú vpravo. Hra sa ukončí kedykoľvek zatvorením jej okna.

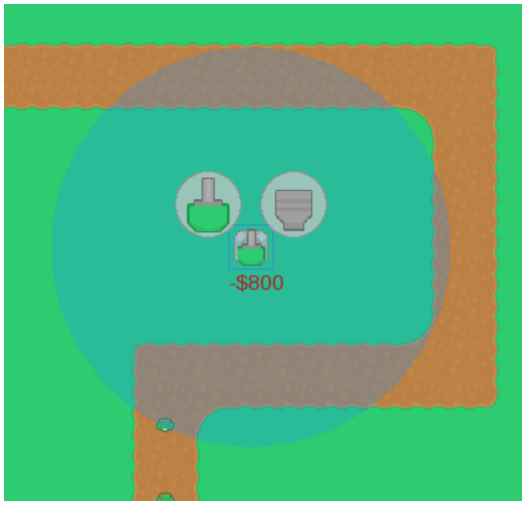

Figure 2: Nákup veží

Veže sa nakupujú kliknutím ľavým tlačidlom myši na voľné políčko. Zobrazia sa dve kruhové tlačidlá. Keď je myš nad niektorým z dvoch tlačidiel, zobrazí sa náhľad na vežu a pod ňou jej cena. Modrý kruh určuje územie, na ktoré má veža dostrel.

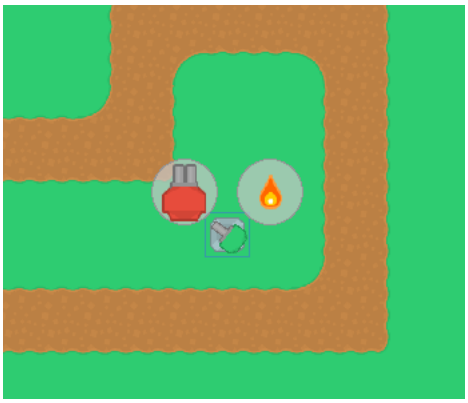

Figure 3: Vylepšenie a predaj veže

Pri kliknutí ľavým tlačidlom myši na nejakú vežu sa zobrazia možné akcie. Každá veža sa dá predať kliknutím na pravé tlačidlo s ikonou ohňa. Niektoré veže majú možnosť vylepšenia, na ktoré slúži ľavé tlačidlo. Taktiež sa zobrazuje náhľad vylepšenej veže a cena vylepšenia.

1.5 Protivníci

Figure 4: Roboty

V hre sa vyskytujú 4 typy robotov, ktorých sila rastie (zľava doprava). Každý robot napravo od iného robota je rýchlejší a má viac zásahových bodov ako jeho ľavý sused. Tiež dáva viac peňazí pri zneškodnení, ale aj uberá viac zásahových bodov hráčovi pri úspešnom prechode.

Drony sa pohybujú rýchlejšie ako roboty. Sivé lietadlo je najsilnejší protivník v hre.

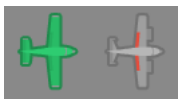

Figure 5: Drony

1.6 Typy veží

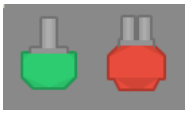

Figure 6: Zelená veža a vylepšená veža

Zelená veža stojí $800. Striela rýchlo, ale nespôsobuje veľké poškodenie. Za $2000 sa dá vylepšiť na červenú vežu, ktorá striela dvakrát tak rýchlo.

Figure 7: Raketomet

Za cenu $4000 je možné zakúpiť raketomet. Má výrazne väčší dosah ako zelená a červená veža. Jeho rakety akcelerujú počas letu. Striela veľmi pomaly, ale spôsobuje veľke škody. Nedá sa vylepšiť.

2 Programátorská príručka

2.1 Vstupný bod a smyčka

Hlavnú úlohu zohráva trieda *App*. Jej metóda *load* je zavolaná práve raz vo vstupnom bode programu. Táto metóda sa postará o načítanie všetkých textúr, fontov a zvukov. Následne metóda loop spustí hernú smyčku (game loop). Trieda *App* pracuje so scénami. Scéna je analógia jednej stránky vrámci webstránky. *App* vlastní najviac jednu inštanciu scény a rieši prechody medzi scénami, keď si to scéna vyžiada.

Game loop sa opakuje kým je okno aplikácie otvorené. Pozostáva z nasledujúcich častí:

1. Prechod medzi scénami, ak si to práve aktívna scéna vyžiadala. *App* predá konštruktoru novej scény parametre z objektu *SceneChangeRequest*.

2. Spracovanie vstupu - eventov. Napríklad kliky myšou, klávesnica apod. Pre každú udalosť - *sf::Event* sa volá metóda *Scene::handleInput(event)*.

   Toto je v aplikácií zaužívaný princíp. Objekty, ktoré vlastnia iné objekty, na nich volajú metódu *handleInput(event)* pre každý obdržaný event.

3. Niekoľko krokov v hernom svete *fixedTimestepUpdate*, používa sa *fixed timestep update*, To znamená, že v každom cykle sa spraví toľko krokov v hernom svete, aby sa za každú sekundu spravilo 30 krokov. Scéna spraví krok po zavolaní *update(delta)*. *Delta* je časová dĺžka, o ktorý sa má svet pohnút dopredu. Keďže robím 30 krokov za sekundu, je to $\frac{1}{30}$ sekundy.

   Tento princíp sa zase opakuje naprieč celou aplikáciou. Rôzne objekty vo svojej metóde *update* volajú *update(delta)* na objekty, ktoré vlastnia.

4. Kreslenie scény. Kresliteľné objekty dedia z triedy *sf::Drawable* a implementujú jej virtuálnu metódu *draw(target, states)* v ktorej vykreslia seba a objekty, ktoré vlastnia.

2.2 Scény

Konkrétne scény sú implementované ako triedy, ktoré dedia zo *Scene*. Scéna implementuje virtuálne metódy *handleInput*, *update* a *draw* popísané vyššie. Volaním metódy *requestSceneChange(request)* si scéna môže vyžiadať zmenu scény.

TextScene je abstraktná scéna, ktorá vie zobraziť 3 texty a po stlačení niektorej klávesy prejde do scény predanej konštruktoru.

Implementované sú nasledujúce scény:

- *WelcomeScene* je scéna so základnými informáciami, ktorá sa zobrazí po spustení aplikácie. Vychádza z *TextScene*, po stlačení klávesy si vyžiada *GameScen*, ktorej nepredáva žiadne informácie.
- *GameScene* je samotná hra. Spravuje objekty užívateľského rozhrania *ContextMenu* a *StatusBar*, herný svet - *World* a *Director*, ktorý riadi prichádzajúce vlny protivníkov. Zmena scény na *GameOverScene* nastane v prípade, že zásahové body hráča klesnú na 0. Predáva sa posledné poradové číslo vlny pod parametrom *waveNumber*.
- *GameOverScene* informuje hráča o *waveNumber*. Vychádza z *TextScene* a po stlačení klávesy nastane prechod do *GameScene*.

2.3 Assets - textúry, fonty a audio

Všetky obrázky sú od autora Kenney Vleugels. Nahrávajú sa ako jeden obrázok vo formáte png - *tilesheet*, ktorý je rozdelený na štvorčeky veľkosti $64px \times 64px$. Využíva sa jediný slobodný font Liberation Sans. Zvukové efekty sú vytvorené v programe Bfxr.

Trieda Assets je singleton držiaci ukazovateľ na *RenderWindow*, inštancie fontu, textúry a *Audio*. *Audio* je dátová štruktúra, ktorá drží načítané zvuky a dokáže efektívne nájsť a prehrať poďla zadaného *SoundEffect*. Singleton sa osvedčil ako lepšia možnosť oproti alternatívam, keďže takmer každý objekt potrebuje mať prístup k aspoň jednej položke zo štvorice textúra, font, zvuk, okno.

2.4 Sektor a mriežka

Terén hry je rozdelený na štvorce veľkosti 64x64 bodov. Polohu tákehoto štvorca (sektoru) reprezentuje trieda *Sector*. Sektory majú vlastný súradnicový systém zložený z dvoch čísel. Sektor v ľavom hornom rohu má súradnice $(0, 0)$, ktoré rastú smerom dole doprava. Niekedy je nutné prejsť od súradníc hry do súradníc sektoru alebo naopak. Na to poskytuje *Sector* metódy *fromCoords*, *upperLeftPoint* apod.

Mriežka *(Grid)* je obdĺžnik zložený zo sektorov, ktoré majú priradenú textúru. Textúra priradená sektoru je reprezentovaná číslom štvorčeku vrámci tilesheetu. Je to rovnaké číslovanie ako využíva program Tiled, teda štvorčeky v prvom riadku zhora sú očíslované zľava doprava 0, 1, ... 22, v druhom riadku 23, 24 ... apod. Toto číslo sa v zdrojových kódoch označuje *textureId*. Mriežka taktiež udržuje cestu - *Path*, čo predstavuje lomenú čiaru vrámci mapy, po ktorej sa hýbu oponenti.

Vlna *(Wave)* predstavuje jednu postupnosť protivníkov rovnakého typu so zadanými rozostupmi. Na požiadanie ich dokáže vygenerovať - metóda *maybeSendNext(delta)* buď vytvorí protivníka umiestniteľného do sveta alebo vráti *nullptr*. Robí to tak, aby boli dodržané dané časové rozostupy.

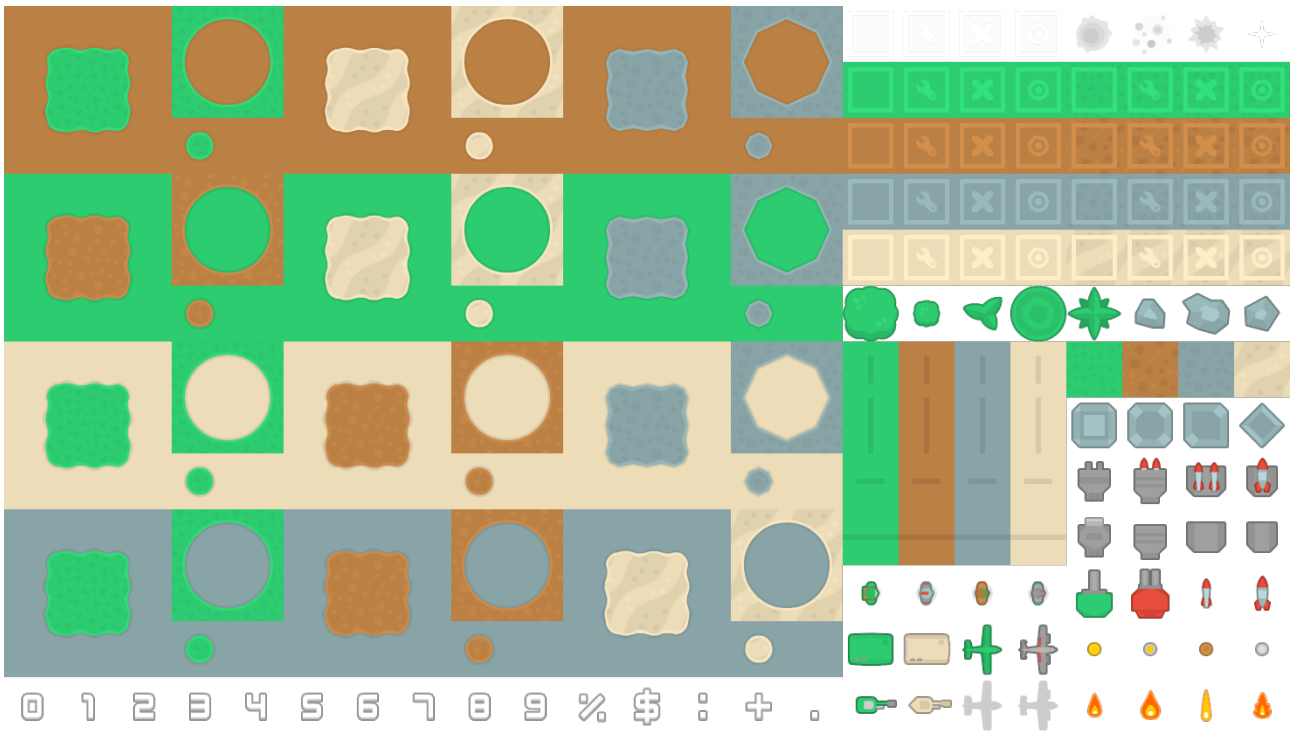

Figure 8: Tilesheet

Director vytvára a riadi vlny protivníkov. Vždy má najviac jednu inštanciu vlny, z ktorej posiela protivníkov do sveta. Keď sa svet vyčistí (neobsahuje žiadneho protivníka), vytvorí si nasledujúcu vlnu. Prvých dvadsiatka vĺn je explictine definovaná, ostatné sa generujú s lineárne sa zvyšujúcim počtom protivníkov.

2.5 Herný svet - World

World je kontajner spájajúci mriežku, entity (veže, projektily, oponenti) a stav (koľko má hráč peňazí a zásahových bodov).

Entita *Entity* je spoločný predok veží, projektilov a oponentov. Vlastní jeden *(sf::Sprite)*, ktorému priradí textúru podľa ID v konštruktore. Ponúka k nemu metódy, ktoré ho dokážu otočiť v smere vektoru alebo posunút so zadanou rýchlosťou pohybu.

Actor predstavuje jedneho oponenta. Je to entita pohybujúca sa po zadanej lomenej čiare. Udržuje si svoje zasáhové body, ktoré mu uberajú projektily. Oponent skončí svoju púť keď prišiel o zásahové body alebo prešiel celú svoju dráhu. *World* odchytáva tieto situácie a príslušnemení svoje stavy. Buď navýši hráčove konto alebo zníži jeho zásahové body o hodnotu *Actor::getWorth()* (respektíve). Odvodené triedy *Soldier* a *Plane* sú konkrétne implementácie oponentov. Sú čisto dátové - len predávajú konkrétne hodnoty konštruktoru *Actor*, ale návrh je pripravený tak, že by mohli obsahovať aj nejaké špecifické správanie.

Projectile je projektil naháňajúci oponenta vo svete. Keď ho chytí, zníži mu zásahové body. V prípade, že oponent stihne zmiznúť pred tým, ako ho projektil chytí, tak zmizne aj projektil. Projektily sú umiestňované do sveta vežami. Implementovaný je *Bullet* a *Rocket*. Tieto sa líšia aj behaviorálne - raketa sa vzhľadom k svojej textúre otáča smerom k cieľu a navyše akceleruje.

Veža *Tower* môže byť hráčom umiestnená na akékoľvek políčko, kde neprechádza cesta. Potom prehľadáva pole oponentov a keď je nejaký v dosahu, teda bližšie ako *Tower::getRange()*, vystrelí naňho projektil. Vystreliť potom môže až keď ubehne čas *Tower::cooldown*. Implementácie veží sú *GreenGun*, *RedTwinGun* a *RocketLauncher*. Líšia sa číslami, ktoré predávajú konštruktoru *Tower* a spôsobom vytvárania projektilov. *RedTwinGun* vyrába projektily tak,

aby sa zdalo že vybiehajú striedavo z ľavej a pravej hlavne. *RocketLauncher* zase striela rakety. Všetky konkrétne veže obsahujú svoje vlastnosti ako verejné statické atribúty. Toto umožňuje použitie metaprogramovania v časti UI.

3 Užívateľské rozhranie

TowerPreview je náhľad veže, ktorá ešte nie je umiestnená vo svete. Využíva sa pred kúpou alebo pred vylepšením veže. Naznačuje kde bude veža umiestená, aký bude mať dostrel a koľko to bude stáť.

Button je všeobecné tlačítko reagujúce na polohu myši a na kliknute. Rozpozná, keď je myš nad ním, a konkrétna implementácia tlačítko sa ho na to môže spýtať volaním *Button::mouseHovers*. Pri kliku zavolá virtuálnu metódu *onClick*. Táto metóda by mala byť zavolaná aj v každej implementácii tlačidla, aby si trieda *Button* správne zaznačila, že tlačidlo už bolo zatlačené.

PlaceTowerButton<T> je tlačítko umiesťnujúce vežu typu *T*. Stará sa zobrazenie náhladu a pri kliku aj o odpočítanie ceny veže a umiestnenie veže do sveta. *UpgradeTowerButton<T>* zase dokáže "vylepšiť" vežu - prakticky zmazať starú vežu, umiestniť na rovnaké miesto novú a odpočítať rozdiel v cenách. *SellTowerButton* vežu odstráni a pripíše hráčovi na konto polovicu pôvodnej ceny.

ContextMenu zobrazuje pod myšou hráča indikátor, aby hráč videl nad ktorým sektorom sa nachádza. Sú dva druhy kontextových menu - nad prázdnym políčkom, kde je možné postaviť vežu, sa zobrazí "shopping list" teda ponuka dvoch tlačítok na kúpenie veže. Toto sú *PlaceTowerButton<GreenGun>* a *PlaceTowerButton<RocketLauncher>*, Nad políčkom, kde je umiestnená veža, máme možnosť predaja - tlačítko *SellTowerButton*, prípadne *UpgradeTowerButton<typVežeNaSektore>*. Sektory cesty sú naznačené červeným indikátorom a nemajú žiadne kontextové menu.