V šľapajach Zorku

Na cvičeniach pracujeme so softvérovým rámcom `Monogame` pre vývoj 2D grafickej hry, v treťom zadaní sa pozrieme na iný typ počítačových hier, a to na textovky. Textová hra pozostáva z niekoľkých kôl, kde používateľ musí zadať príkazy do konzoly počítača a hra na jeho kroky zareaguje výpisom, kde je mu vysvetlené, čo sa v hre udialo. V zadaní budete ovládať princeznú, ktorá sa musí prebojovať prekážkami a príšerami a vyslobodiť začarovaného princa. Okay, v pôvodnej forme je to síce žaba, ale po bozku od pravej lásky sa určite premení na rozprávkového princa.

Kostru riešenia nájdete v predpripravenom projekte. Keďže vaše riešenia prejdú automatizovanými testami, je dôležité, aby ste dodržali štruktúru projektu, najmä čo sa týka menných priestorov. Ak programujete v IDE, ktoré vytvára inú štruktúru ako vidíte v kostre riešenia, nezabudnite upraviť štruktúru pred finálnym odovzdaním.

Všetky členské premenné musia byť privátne, a nesmiete k nim pristupovať priamo mimo triedy. Členské premenné v nadtriedach môžu byť `protected`. Vaše riešenie môžete rozšíriť o premenné a metódy okrem tých uvedených v zadaní. Projekt obsahuje veľký počet tried, niektoré sú plne pripravené, v iných potrebujete doplniť funkcionalitu. Konštruktory sú vždy implementované tak, aby postačili na ďalšiu implementáciu, ak ale chcete do nich niečo pridať, je to možné (pôvodnú funkcionalitu však nemeňte).

Herné objekty

V hre sa nachádza niekoľko rôznych herných objektov, ktoré zastrešujú rôzne akcie, ktoré hráč môže vykonávať. Všetky objekty ako aj podporné triedy a rozhrania sú definované v mennom prostredí `GameObjects`. V priečinku nájdete dve abstraktné triedy, ktoré sú plne implementované a zabezpečujú všeobecnú funkcionalitu.

`AbstractObject` je základná trieda, od ktorej priamo alebo nepriamo dedia všetky triedy definujúce herné objekty. Každý objekt je popísaný menom (`name`) – bude jedinečný identifikátor – a krátkym popisom (`description`). Okrem toho trieda uchováva referenciu na miestnosť (`currentRoom`), v ktorej sa objekt aktuálne nachádza (zároveň zoznam prítomných objektov bude uchovávaný aj v triede `Room`). Okrem konštruktora, ktorý inicializuje hodnoty členských premenných v triede nájdete aj metódu `AddToRoom(Room room)`, ktorá nastaví referenciu na aktuálnu miestnosť, ako aj gettery pre jednotlivé členské premenné. Trieda definuje ešte metódu `Update()`, ktorá má implementovať správanie každého objektu v jednotlivých herných kolách. V triede `AbstractObject` ostáva prázdna (nie je abstraktná z dôvodu, že niektoré podtriedy túto metódu nebudú prepisovať a tak by sme mali zbytočne veľa prázdnych implementácií metódy).

Jednou podtriedou `AbstractObject` je `AbstractActor`, ktorá predstavuje postavu v hre, ktorá má život (vždy sa nastavuje na 100 na začiatku). Trieda doimplementuje metódu pre zmenu života (`ChangeHealth(int delta)`), ako aj na zistenie toho, či postava stále žije (`IsAlive()` – hodnota života je viac ako 0).

Okrem toho v priečinku nájdete rozhrania:

- `IItem` – marker rozhranie pre objekty, ktoré môžu byť vložené do batoha hráča;

- `IPrince` – rozhranie deklarujúce funkcionalitu pre princa, teda možnosť pobozkať ho (bude definovaná pre žabiu ako aj ľudskú formu);
- `IUsable` – rozhranie deklarujúce použiteľné objekty s dvomi metódami: `Use()` pre použitie objektu a `WasUsed()` pre zistenie toho, či už bol objekt použitý (v prípade že má limitovaný počet použití).

Do projektu doimplementujte niekoľko herných objektov s nasledovnou funkcionalitou (popis hlavnej postavy `Princess` bude uvedený neskôr):

`Torch`
Trieda `Torch` reprezentuje fakľu, ktorá pomôže princeznej poobzerať sa v tmavých miestnostiach a nájsť objekty, ktoré sa v nej nachádzajú. Trieda dedí od `AbstractObject` a implementuje rozhrania `IItem` a `IUsable`. Okrem zdedených členských premenných v triede nájdete premenné:
- `wasUsed` (`bool`) – vyjadruje, či už bola fakľa použitá – môže sa použiť iba raz, potom sa zhasne;
- `isActive` (`bool`) – vyjadruje, či fakľa práve svieti;
- `running` (`int`) – uchováva počet kôl od momentu keď sme fakľu zapálili – túto hodnotu potrebujeme z dôvodu, že fakľa horí a svieti iba limitovaný počet kôl.

V triede implementujte metódy:
- `Use()` – predpísaná metóda z rozhrania `IUsable`, reprezentuje moment zapálenia fakle, nastavte príslušné členské premenné. Nezabudnite skontrolovať, či už fakľa nebola použitá – v tomto prípade sa nič nevykoná.
- `WasUsed()` – metóda vráti, či môže byť fakľa ešte použitá a zapálená.
- `Update()` – metóda aktualizuje počítadlo `running` ak je fakľa zapálená. Ak fakľa už horí maximálny počet kôl – v našej hre sú to tri kolá –, tak sa fakľa zhasne, a ostane nepoužiteľnou.

`Sword`
Trieda definuje meč, ktorý bude jediným pomocníkom princeznej proti všelijakým príšerám. Dedí od triedy `AbstractObject` a implementuje rozhrania `IItem` a `IUsable`. V konštruktore sa nastaví hodnota dodatočnej členskej premennej `damage`, ktorá vyjadruje hodnotu, o koľko použitie meča zníži život nepriateľa.

V triede implementujte metódy:
- `Use()` – predpísaná metóda z rozhrania `IUsable`, reprezentuje použitie meča. Nájdite v aktuálnej miestnosti nepriateľa (vždy drak s menom `Smaug`) a keď ho nájdete, aktualizujte jeho život na základe udeleného zranenia mečom.
- `WasUsed()` – keďže meč nemá limitovaný počet použití, vracia stále `false`.

`Dragon`
Trieda `Dragon` reprezentuje nášho hlavného nepriateľa, draka ktorý stráži začarovaného princa. Keďže drak je živá bytosť, trieda dedí od `AbstractActor`.

V triede implementujte metódy:
- `ChangeHealth()` – rozšírte základnú implementáciu tak, aby sa drak vymazal zo sveta ak jeho život dosiahne hodnotu 0.

- `Update()` – ak sa v miestnosti draka ocitne princezná, drak na ňu zaútočí, t.j. zníži jej život o 20.

`Spikes` a `Lever`

Trieda `Spikes` predstavuje hroty, ktoré našu princeznú zabijú okamžite ak do nich vstúpi. Našťastie však môžu byť jednoducho zasunuté do stien miestnosti a tak zneškodnené – stačí aktivovať páku `Lever`, ktorá ich riadi a nachádza sa v inej (bezpečnej) miestnosti.

Trieda `Spikes` dedí od `AbstractObject` a rozširuje ju booleovskou hodnotou `active`, ktorá vyjadruje či sú hroty vysunuté a tak predstavujú hrozbu princeznej.

V triede implementujte metódy:
- `Toggle()` – prepína aktiváciu hrotov (zmeňte hodnotu členskej premennej `active`).
- `Update()` – ak hroty nie sú vysunuté resp. aktívne, nič sa nestane. Ak hroty sú aktívne a princezná sa nachádza v rovnakej miestnosti, hroty ju zrania o 100, teda ju určite zabijú.

Trieda `Lever` rovnako dedí od `AbstractObject`, predstavuje však použiteľný objekt a tak implementuje rozhranie `IUsable`. Podobne ako `Spikes` definuje členskú premennú `active` pre uchovávanie informácie o stave zapnutia páky resp. hrotov. Referenciu na hroty, ktoré riadite pákou nájdete v členskej premennej `spikes` (typu `Spikes`).

V triede implementujte metódy:
- `ConnectSpikes(Spikes spikes)` – napojí páku na hroty, a teda nastaví referenciu. Dbajte na to, aby po prepojení týchto dvoch objektov ostali oba aktívne.
- `Use()` – slúži na prepínanie stavu páky, a podľa toho aktualizuje aj stav hrotov.
- `WasUsed()` – počet použití páky nie je limitovaný, vždy vracia `false`.

`FrogPrince` a `HumanPrince`

Triedy `FrogPrince` a `HumanPrince` slúžia na reprezentáciu rôznych foriem začarovaného princa (žaba a človek), pričom obe triedy dedia od `AbstractActor` (aj keď život princa nikdy nebudeme aktualizovať) a implementujú rozhranie `IPrince`. V triedach potrebujete implementovať iba metódu `Kissed()` s nasledovnou funkcionalitou:
- v triede `FrogPrince` ak princezná pobozká žabu, tá sa premení na ozajstného princa, teda z aktuálnej miestnosti odstráňte žabieho princa a pridajte princa človeka s rovnakým menom ako meno žaby.
- v triede `HumanPrince` pobozkanie princa znamená výhru, nastavte teda premennú pre reprezentáciu výherného stavu v hernom svete (bude popísaná neskôr).

`Cage` a `Key`

Nestačí, že princ bol začarovaný, bol aj zavretý do klietky (`Cage`), z ktorej ho musíme vyslobodiť. Našťastie sa vo svete nachádza aj kľúč (`Key`), ktorý otvorí akúkoľvek klietku.

Trieda `Cage` dedí od triedy `AbstractObject` a rozširuje ju členskými premennými:
- `locked` (`bool`) – vyjadruje, či je klietka zavretá.
- `captive` (`IPrince`) – princ (v akejkoľvek forme) uväznený v klietke.

V triede implementujte nasledovnú funkcionalitu:

- v konštruktore inicializujte členské premenné – klietka bude zavretá a bude uchovávať princa v žabej podobe s menom *prince* a popisom *enchanted prince in frog form* (objekt princa musíte vytvoriť priamo v konštruktore).
- `Open()` – metóda otvorí klietku a pridá do aktuálnej miestnosti uväzneného princa.

Trieda `Key` dedí tiež od triedy `AbstractObject`, môžeme si ho ale zobrať so sebou aj ho použiť, takže implementuje rozhrania `IItem` a `IUsable`. Do triedy potrebujete doplniť iba metódy súvisiace s rozhraním `IUsable`:
- `Use()` – použitie kľúča znamená, že ak sa v aktuálnej miestnosti nachádza klietka, tak ju otvoríme.
- `WasUsed()` – kľúč môžeme použiť koľkokrát len chceme, stále vracia `false`.

Herný svet
Teraz už máme pripravené všetky herné objekty, ale samozrejme sa celá hra musí odohrávať niekde. V zadaní konkrétne sa jedná o miestnosti (dva typy), a množina prepojených miestností reprezentuje herný svet. Všetky koncepty potrebné pre reprezentáciu herného sveta nájdete v mennom priestore `GameWorld`. Nachádza sa tam aj jedna pomocná enumerácia definujúce možné smery pohybu resp. prepojenia miestností – `Directions` s hodnotami `NORTH`, `EAST`, `SOUTH` a `WEST`.

`Room` a `DarkRoom`
Miestnosť bude reprezentovaná triedou `Room`, pričom podobne ako herné objekty aj miestnosti budú mať meno a popis a ďalšie pomocné členské premenné:
- `name` (`string`) – meno miestnosti;
- `description` (`string`) – podrobnejší popis miestnosti;
- `world` (`World`) – herný svet, ktorého súčasťou je aj aktuálna izba;
- `north`, `south`, `east`, `west` (`Room`) – susediace miestnosti (ak také existujú) v jednotlivých dovolených smeroch;
- `objects` (`List<AbstractObject>`) – zoznam herných objektov, ktoré sa nachádzajú v miestnosti.

V triede potrebujete implementovať nasledovné metódy:
- `AddToRoom(AbstractObject obj)` – metóda pridá herný objekt do miestnosti, ak ten sa v nej ešte nenachádza (predíďte duplikátom). Okrem aktualizáciu zoznamu herných objektov nezabudnite aktualizovať aj referenciu na aktuálnu miestnosť v samotnom hernom objekte.
- `RemoveFromRoom(AbstractObject obj)` – metóda odstráni z miestnosti herný objekt, samozrejme iba ak tam vôbec je.
- `AddNeighbor(Room room, Directions direction)` – metóda pridá susediacu izbu do príslušnej členskej premennej na základe druhého argumentu.
- `GetObjectWithName(string name)` – metóda vráti herný objekt so zadaným menom, ak sa ten nachádza v miestnosti, v opačnom prípade vráti `null`. Pri prehľadávaní veľkosť písmen neberte do úvahy.
- `GetObjectNames()` – metóda vráti zoznam mien všetkých objektov, ktoré sa nachádzajú v miestnosti.
- `GetDirections()` – metóda vráti zoznam smerov, v ktorých sa môžeme presunúť z danej izby (v danom smere existuje susediaca izba). Možné hodnoty v zozname sú *north*, *east*, *south* a *west*.

- `GetNeighbor(Directions direction)` – metóda vráti referenciu na susediacu izbu v zadanom smere, alebo `null` ak taká izba neexistuje.

V triede `DarkRoom`, ktorá predstavuje tmavú izbu, v ktorej vidíme iba v prípade ak tam svieti fakľa, upravte implementáciu metódy `GetObjectNames()` tak, aby vrátila zoznam mien prítomných objektov iba v prípade, ak sa v miestnosti nachádza svietiaca fakľa. V opačnom prípade metóda vráti prázdny zoznam (nie `null`).

`ObjectFactory`
Trieda `ObjectFactory` slúži na vytváranie herných objektov na základe načítaných údajov (využitie pri načítavaní herných svetov). V triede potrebujete implementovať iba metódu `CreateObject`, ktorá vytvorí inštanciu správneho typu na základe zadaných parametrov. Možné triedy sú `Cage`, `Dragon`, `Key`, `Lever`, `Princess`, `Spikes`, `Sword`, `Torch`. Ak príde nesprávny typ, vygenerujte chybu `ArgumentException` so správou *Unknown actor type XY*, kde namiesto *XY* vypíšete zadanú hodnotu `actorType`.

Poznámka: V triede `Lever` nastavte referenciu na `Spikes` na `null`. Hodnotu `damage` pre triedu `Sword` nastavte na hodnotu 50.

`World`
Trieda `World` reprezentuje herný svet a rieši spustenie hry. V triede nájdete členské premenné:
- `map` (`List<Room>`) – zoznam (prepojených) miestností s pridanými objektmi, čiže samotný svet;
- `won` (`bool`) – booleovská hodnota vyjadrujúca, či hráč vyhral;
- `done` (`bool`) – booleovská hodnota vyjadrujúca, či sa hra skončila (výhrou alebo prehrou);
- `lost` (`bool`) – booleovská hodnota vyjadrujúca, či hráč prehral;
- `factory` (`ObjectFactory`) – pomocný objekt pre vytváranie herných objektov.

V triede sú implementované nasledovné metódy:
- konštruktor nastaví iniciálnu hodnotu členských premenných.
- `SetWin()`/`SetDone()`/`SetLoss()` nastavia hodnotu príslušných členských premenných na `true`.
- `RunGame()` rieši spustenie a priebeh hry, realizuje jednotlivé herné kolá až dovtedy, kým sa hra neskončí výhrou, prehrou alebo vypnutím.

Do triedy pridajte implementáciu metódy `LoadMap(string mapPath)`, ktorá načíta herný svet na základe štruktúrovaného JSON súboru, cestu ku ktorému dostanete ako argument metódy. Na základe týchto údajov najprv vytvorte jednotlivé miestnosti, následne ich prepojte, pridajte do nich herné objekty, a ak je potrebné, tieto herné objekty prepojte (pre `Spikes` a `Lever`). Štruktúru JSON objektu nájdete popísanú v osobitnej príručke, ukážkový JSON súbor s názvom `sample_map.json` nájdete v projekte.

`Princess` a akcie
Interakcia s hrou a zadávanie príkazov sú zabezpečené cez triedu `Princess`, ktorá zároveň reprezentujú hlavnú postavu hry. V metóde `Update()` sa načíta príkaz od používateľa, a následne sa spracuje priamo v triede. Vykonávanie príkazov budú zastrešovať osobitné triedy

reprezentujúce jednotlivé možné príkazy, ktoré sú definované v mennom priestore `GameActions`.

Trieda `Princess` dedí od triedy `AbstractActor` a definuje jednu dodatočnú premennú `backpack` typu `IItem`, ktorá reprezentuje batoh hráča, do ktorého môže vložiť rôzne objekty (naraz však len jeden). Na začiatku je batoh inicializovaný ako prázdny, neskôr je možné do neho pridať predmety.

V triede implementujte nasledovné metódy:

- `AddToBackPack(IItem toBeAdded)` – pridá do batoha objekt, ktorý dostane ako argument. Zároveň ak sa v batohu nachádzal objekt, ten musíte pridať do aktuálnej miestnosti.
- `EmptyBackPack()` – vyprázdni batoh a objekt, ktorý sa v ňom nachádza, pridá do aktuálnej miestnosti hlavnej postavy. Ak batoh je prázdny, nič sa neudeje.
- `ChangeHealth(int delta)` – upravte metódu tak, že ak princeznej dôjde život, hru prehráme (aktualizujte príslušnú premennú v triede `World`).
- `MoveToRoom(Room newRoom)` – metóda rieši presun princeznej (a predmetu v batohu) do novej miestnosti. Nezabudnite potrebné objekty tiež odstrániť z aktuálnej miestnosti. Objekt uložený v batohu **nemá byť** v zozname prítomných objektov miestnosti.
- `ProcessInput(string userInput)` – metóda spracuje príkaz hráča, ktorý načítame v metóde `Update()`. Pre každý príkaz vytvorte príslušný objekt (popísané nižšie) a vykonajte akciu. Ak akciu neviete pripraviť pretože hráč zadal nesprávny príkaz, vypíšte správu *I do not understand what you mean*. Príkazy a ich štruktúra sú popísané nižšie.

Akcie implementujte v osobitných triedach v priečinku `GameActions`. Každá trieda implementuje rozhranie `IAction` s povinnou metódou `Execute(AbstractActor actor)`, kde parameter reprezentuje postavu, ktorá chce vykonať danú akciu (v našej hre to reálne bude stále princezná). Štruktúra tried je už pripravená, v každej z nich potrebujete implementovať iba metódu `Execute()`.
Hráč môže pracovať s nasledovnými akciami:

- `Exit` (príkaz je *exit*) – vypne hru, nastavte hodnotu členskej premennej done v triede `World`.
- `CheckBag` (príkaz je *check bag*) – vypíše do konzoly meno objektu, ktorý princezná má v batohu. Ak parameter metódy nie je princezná, nič sa nevykoná, rovnako sa nič nestane ak jej batoh je prázdny. Výpis riešte správou *The princess has XY in her backpack*, kde *XY* nahradíte menom objektu v batohu.
- `Inspect` (príkaz je *inspect XY*, kde *XY* je meno objektu) – preskúmanie objektu, pričom sa do konzoly vypíše podrobný popis (`description`) daného objektu. Objekt, ktorý chcete preskúmať dostane samotný konštruktor (samozrejme ak objekt existuje). Ak hráč chce preskúmať objekt, ktorý sa nenachádza v aktuálnej miestnosti alebo batohu princeznej, príkaz nebude spracovaný a vypíše sa chybová správa *I do not understand what you mean*.
- `Kiss` (príkaz je *kiss XY*, kde *XY* je meno objektu) – princezná pobozká princa (v akejkoľvek forme). Objekt princa, ktorý chcete pobozkať dostane samotný konštruktor (samozrejme ak existuje). Ak princ so zadaným menom neexistuje v aktuálnej miestnosti príkaz nebude spracovaný a vypíše sa chybová správa *I do not understand what you mean*.

- `LookAround` (príkaz je *look around*) – slúži na preskúmanie miestnosti a získanie bližších informácií o nej. Do konzoly sa určite vypíše podrobný popis miestnosti (`description`) a zoznam objektov, ktoré sa nachádzajú v miestnosti. Ak hráč nič nevidí v miestnosti, vypíšte *You don't see anything*. V opačnom prípade vypíšte *You see: XY AB CD*, kde medzerou oddelené budete mať mená objektov. Dbajte na to, aby princezná a objekt v jej batohu neboli vypísané.
- `Move` (príkaz je *move north/east/south/west*) – umožňuje presun princeznej do inej miestnosti (ak v danom smere existuje susediaca miestnosť). Smer dostanete ako parameter konštruktora. Ak hráč zadá nesprávny príkaz (napríklad *move southeast*), vypíšte chybovú správu *I do not understand what you mean*. Ak v zadanom smere neexistuje susediaca miestnosť, vypíšte *I cannot go there*. Ak sa hráč môže presunúť v danom smere, tak princeznú (a prípadne objekt v jej batohu) presuňte do príslušnej novej miestnosti. Ak sa však v aktuálnej miestnosti nachádza drak, ten znemožní presun princeznej do akejkoľvek miestnosti.
- `PickUp` (príkaz je *pick up XY*, kde *XY* je meno objektu) – princezná si zoberie daný predmet z aktuálnej miestnosti. Ak hráč zadá nedostupný objekt, alebo objekt, ktorý nie je presúvateľným predmetom, vypíšte chybovú správu *I do not understand what you mean*. Ak je príkaz správny, objekt pridajte do batoha hráča a odstráňte ho z aktuálnej miestnosti. Ak princezná niečo už má v batohu, vypíšte správu *My backpack is full*.
- `PutDown` (príkaz *put down*) – príkaz vyberie objekt z batoha princeznej a pridá ho do aktuálnej miestnosti. Ak princeznin batoh je prázdny, vypíšte chybovú správu *There is nothing in my backpack*.
- `Use` (príkaz *use XY*, kde *XY* je meno objektu) – umožní použitie objektu. Ak zadaný objekt nie je dostupný (nie je v miestnosti alebo batohu princeznej), alebo nie je použiteľný, vypíšte chybovú správu *I do not understand what you mean*. V opačnom prípade vykonajte použitie objektu (objekt musíte odovzdať priamo v konštruktore akcie).

Príklad definície herného sveta nájdete v súbore `sample_map.json`. Jeho štruktúra je opísaná v osobitnom súbore.

Hodnotenie zadania:
- `Room` a `DarkRoom` – spolu 1 bod
- `World` – 1 bod (iba `LoadMap()`)
- `ObjectFactory` – 0,5 bodov (iba `CreateObject()`)
- `Cage`, `FrogPrince`, `HumanPrince`, `Key`, `Spikes`, `Sword` – po 0,3 body
- `Dragon`, `Lever`, `Torch` – po 0,4 body
- `Princess` – 1,5 boda
- všetky akcie okrem `Move` – po 0,3 body
- `Move` – 0,6 bodov