1. Actor – klasa abstrakcyjna – reprezentacja obiektów (aktorów) występujących w pomieszczeniach.
    a. Atrybuty:
        - equipement, List<Item> ~protected – reprezentacja stanu posiadania danego aktora.
    b. Metody:
        - Examine(), String ~public abstract – zwraca tekst opisu aktora.
        - Name(), String ~public abstract – zwraca tekst z nazwą aktora.
        - BeEquiped(), void ~public virtual – jeżeli aktor jest obiektem typu Item (bądź dziedziczącego po Item), to zwraca siebie samego w postaci obiektu typu Item.
        - GetLoot(), List<Item> ~public – zwraca pole equipement
2. Beholder – kontroler w modelu MVC – przetwarza informacje od użytkownika i na podstawie ich przekazuje komendy modelowi oraz widokowi.
    a. Atrybuty:
        - GO, EXAMINE, TAKE, HIT, SURRENDER, KIDDING, EQUIPEMENT, FORWARD, BACK, LEFT, RIGHT, UP, DOWN, HELP, commandList – pomocnicze zmienne typu string zapisane w celu ułatwienia zarządzania komendami.
        - playground, Dungeon ~private – wskaźnik na model.
        - storyteller, Fiddler ~private – wskaźnik na widok.
        - goal, Func<Dungeon, bool> ~private – predykat wskazujący, czy zadany cel gry został spełniony.
    b. Metody:
        - PlayTheGame(), void ~public – metoda zarządzająca komendami użytkownika odpowiednio, na ich podstawie, przekazująca informacje modelowi i widokowi.
        - End(), string ~private – zwraca tekst towarzyszący przegranej i wykonuje wszelkie działania w tle z tym związane.
        - Surrender(), string ~private – zwraca tekst towarzyszący poddaniu się i wykonuje wszelkie działania w tle z tym związane.

3. Creature : Actor – klasa abstrakcyjna – reprezentacja istot występujących w pomieszczeniach.
    a. Atrybuty:
        - hp, int ~protected – reprezentacja poziomu zdrowia istoty.
        - ad, int ~protected readonly – reprezentacja ilości zadawanych obrażeń przez istotę.
    b. Metody:
        - Damage(int damage), void ~public – odbiera zadaną przez damage liczbę punktów zdrowia istocie.
        - IsAlive(), bool ~public – predykat zwracający odpowiedź na pytanie, czy hp istoty jest większe od zera.
        - MakeAction(Hero wanderer, Room hall), string ~internal abstract – pole, przygotowane pod metodę wykonania akcji w danej turze przy zadanym bohaterze oraz pomieszczeniu, w którym istota się znajduje, zwracające tekst opisujący wykonaną akcję.
4. Direction – klasa statyczna – reprezentacja kierunków świata.
    a. Atrybuty:
        - FORWARD, BACK, LEFT, RIGHT, UPSTAIRS, DOWNSTAIRS, int ~public const – stałe reprezentujące poszczególne kierunki.
    b. Metody:
        - Turn(int face, int rotation), int ~public static – za argumenty przyjmuje aktualny zwrot oraz kierunek względny, w którym zamierzony jest obrót oraz zwraca przetworzony zwrot.
        - Reverse(int direction), int ~public static – przyjmuje kierunek i zwraca mu przeciwny.
5. Door – klasa będąca reprezentację przejść pomiędzy pomieszczeniami.
    a. Atrybuty:
        - inside, outside, Room ~private readonly – wskaźniki na pokoje po obu stronach drzwi.
        - locked, bool ~private – informacja, czy drzwi są zamknięte.
        - key, Key ~private readonly – wskaźnik na klucz, który otwiera te drzwi. Null, jeśli drzwi są zawsze otwarte.
    b. Metody:
        - Door(Room inside, Room outside, Key key, int direction) ~public – konstruktor, który, oprócz przypisania pól przypisuje siebie na wskaźniki w odpowiednich pokojach na podstawie informacji direction.
        - GoThrough(Room from), Room ~public – na podstawie argumentu from zwraca odpowiednie pomieszczenie lub null, jeśli argument nie jest pomieszczeniem z tymi drzwiami lub drzwi są zamknięte.
        - Unlock(Item key), bool ~public – zwraca prawdę, jeśli drzwi zostały odblokowane (wcześniej, lub za pomocą podanego klucza), lub fałsz, jeżeli pozostają zamknięte.

6. Dungeon – model w MVC, odpowiada za wewnętrzną mechanikę gry.
    a. Atrybuty:
        - hall, Room ~private – wskaźnik na pokój, w którym aktualnie dzieje się rozgrywka.
        - wanderer, Hero ~private – wskaźnik na sterowanego bohatera.
        - generator, Func<Room> ~private readonly – bezargumentowa funkcja generująca świat gry.
    b. Metody:
        - Go(int direction), string ~internal – przemieszcza rozgrywkę do wskazanego pomieszczenia lub pozostaje w miejscu, jeśli nie ma możliwości przejścia oraz zwraca tekst o tym opowiadający.
        - Examine(int index), string ~internal – zwraca wynik wywołanej na aktorze o numerze index metody Examine() lub informację, o braku takiego aktora.
        - ExamineThisHall(), string ~internal – zwraca informację o wszystkich drzwiach i aktorach w tym pomieszczeniu.
        - Take(int index), string ~internal – jeśli istnieje dany przedmiot i istnieje możliwość podniesienia go, to przemieszcza go do ekwipunku gracza. Zwraca raport o tej czynności.
        - Hit(int index), string ~internal – nakazuje bohaterowi uderzenie danej istoty (lub powietrza) oraz zwraca z tego raport.
        - MakeTurn(), string ~internal – wywołuje na wszystkich istotach w pomieszczeniu metodę MakeAction(this.wanderer, this.hall) oraz zdaje z tego raport.
        - Rebuild(), void ~public – tworzy podziemia od nowa.
7. Ezgara : Creature – klasa reprezentująca jeden z rodzajów stworzeń występujących w podziemiach.
    a. Atrybuty:
        - stolen, bool ~private – informuje, czy Ezgara ukradł właśnie graczowi jakiś przedmiot i stara się z nim uciec.
    b. Metody:
        - MakeAction(Hero wanderer, Room hall), string ~internal override – przy wokonywaniu akcji próbuje okraść gracza z losowego przedmiotu, gdy mu się to uda zaś stara się uciec z pomieszczenia wraz ze skradzionym przedmiotem.
        - Run(Room hall), bool ~private – metoda znajdująca losowe pomieszczenie, do którego może uciec Ezgara. W zależności od wyniku losowania Ezgara tam ucieka lub nie udaje się to i krąży po pomieszczeniu, co jest zwracane w postaci wartości bool.

8. Fiddler – widok w MVC.
    a. Atrybuty:
        - last, string ~private – ostatni fragment powieści opowiadany przez Skrzypka.
    b. Metody:
        - BeginStory, string ~internal – zwraca tekst rozpoczynający grę.
        - Tell(string v), void ~internal – wyświetla po wyczyszczeniu konsoli wiadomość v.
        - Hint(string v), void ~internal – dopowiada wiadomość v bez czyszczenia konsoli.
        - Repeat(), void ~internal – powtarza ostatni tekst po wyczyszczeniu konsoli.
        - EndWith(string v), void ~internal – wyświetla komunikat końcowy po wyczyszczeniu konsoli. Przygotowana pod ewentualne dodatkowe czynności przy zakończeniu gry.
9. Gulden : Item – klasa reprezentująca jeden z rodzajów przedmiotów występujących w grze, złotą monetę. Klasa ta uzupełnia tylko metody abstrakcyjne Examine() i Name().
10. Hero : Actor – klasa reprezentująca bohatera sterowanego przez gracza.
    a. Atrybuty:
        - strength, agility, hp, int ~private – statystyki bohatera.
        - face, int ~private – zwrot bohatera.
        - name, string ~private – imię bohatera.
        - BASICHP, int ~public static – domyślna wartość punktów życia bohatera.
    b. Metody:
        - Damage(int damage), void ~public – odejmuje damage punktów zdrowia od punktów zdrowia bohatera.
        - Equip(Item item), void ~public – dodaje przedmiot item do ekwipunku bohatera.
        - Turn(int direction), void ~public – zmienia zwrot bohatera o kierunek direction.
        - IsAlive(), bool ~public – predykat sprawdzenia, czy hp > 0.
        - GoForward(Room location), Room ~public – zwraca pomieszczenie za drzwiami naprzeciwko lub null, gdy drzwi są zamknięte.
        - Bagpack(), string ~internal – zwraca raport o zawartości ekwipunku gracza.
        - Reborn(), void ~public – przywraca wszystkie statystyki do stanu początkowego.

11. Item : Actor – klasa abstrakcyjna reprezentująca przedmioty znajdujące się w pomieszczeniach. Jedynym polem zmienionym w tej klasie jest metoda BeEquiped, która zwraca wskaźnik na sam obiekt.
12. Jawler : Creature – klasa reprezentująca jeden z rodzajów stworzeń występujących w pomieszczeniach. Klasa implementuje metodę MakeAction(Hero wanderer, Room hall), każdej tury szczękacz zadaje określoną ilość obrażeń bohaterowi.
13. Key : Item – klasa reprezentują jeden z rodzajów przedmiotów występujących w grze. Klasa ta uzupełnia metody Examine() i Name().
14. LevelGenerator – klasa pomocnicza służąca do generowania poziomów i celów.
    a. Metody:
        - GenericLevel(), Room ~private – zwraca główne pomieszczenie świeżo wygenerowanego poziomu.
        - GenericGenerator(), Func<Room> ~public – zwraca lambda wyrażenie zwracające wynik metody GenericLevel().
        - GenericGoal(), Func<Dungeon, bool< ~public – zwraca lambda wyrażenie będące predykatem informującym o spełnieniu celu w zależności od stanu świata.
15. Locker : Actor – klasa reprezentująca skrzynie znajdujące się w pomieszczeniach.
    a. Atrybuty:
        - key, Key ~private readonly – wskaźnik na klucz otwierający skrzynię.
        - locked, bool ~private – informuje, czy skrzynia jest zamknięta.
    b. Metody:
        - Unlock(Key key), bool ~public - zwraca prawdę, jeśli skrzynia została odblokowana (wcześniej, lub za pomocą podanego klucza), lub fałsz, jeżeli pozostaje zamknięta.
16. Painting : Actor – klasa reprezentująca malowidła występujące w pomieszczeniach. Rozwija metody Examine() i Name().
17. Program – klasa główna programu, rozpoczyna grę.
18. Room – klasa reprezentująca pomieszczenia w grze.
    a. Atrybuty:
        - staticActors, List<Actor> ~private – lista statycznych aktorów w pomieszczeniu.
        - dwellers, List<Creature< ~private – lista stworzeń występujących w pomieszczeniu.
        - north, south, east, west, upstairs, downstairs, Door ~private – wskaźniki na drzwi w konkretnych kierunkach.
    b. Metody:
        - GoTo(int where), Door ~public – zwraca drzwi w zadanym kierunku.
        - Examine(int face), string ~public – zwraca opis tego, co widzi bohater skierowany w daną stronę.

Sterowanie:

- go {forward, back, right, left, up, down} – przejście przez konkretne drzwi
- examine {0…} – sprawdzenie konkretnego obiektu.
- take {0…} – podniesienie konkretnego przedmiotu.
- hit {0…} – uderzenie konkretnego stworzenia.
- surrender – zakończenie rozgrywki.
- kidding – rozpoczęcie rozgrywki od nowa.
- eq – wyświetlenie zawartości ekwipunku.