LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

Mikael Sjödin, miksj784@student.liu.se
Jonas Malmlöf, jonma993@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2016
Version 1.1

2016-12-16

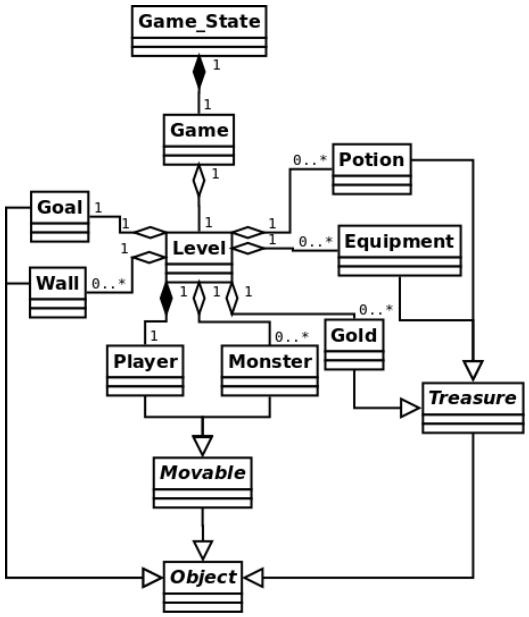
1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.0	Skapade dokumentet och skrev det första utkastet	161125
1.1	Uppdaterade dokumentet så att det stämmer med den faktiska designen	161216

2 Designbeskrivning

2.1 Klassförhållanden

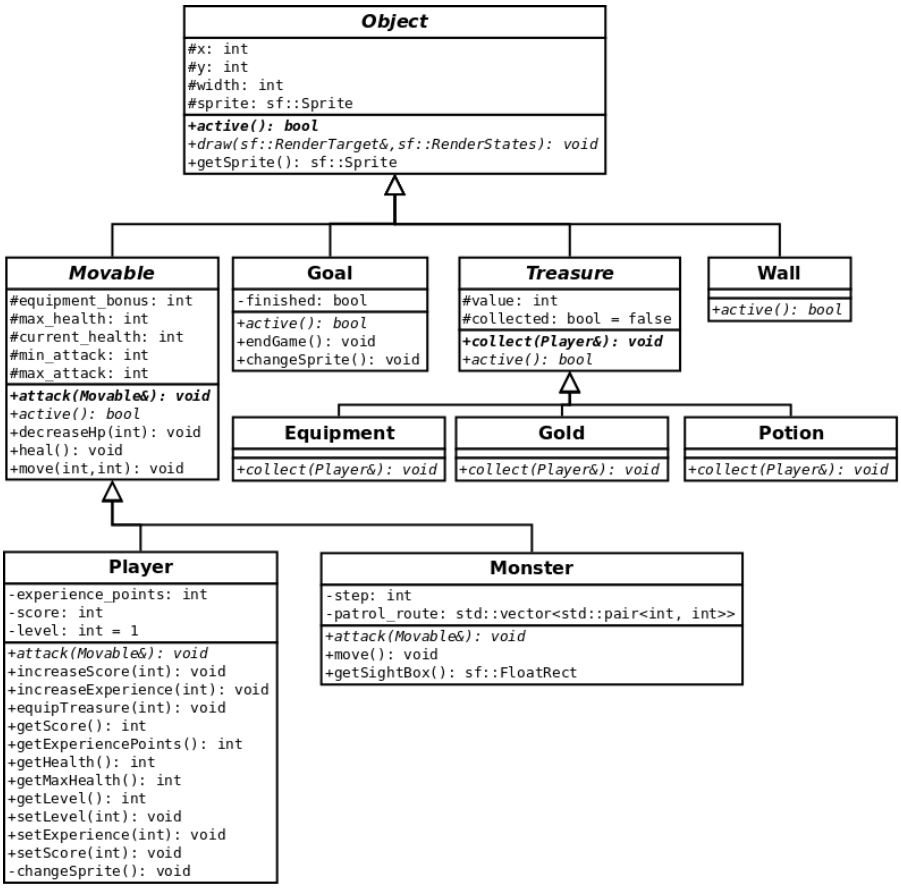
Vår design bygger på att main-funktionen skapar en Game_State och sedan kör denna. Fördelen med detta designval att ha klassen Game_State som ett mellansteg till spellogikens Game, snarare än att anropa Game direkt från spelets main-funktion, är att det blir relativt enkelt att implementera fler spelstadier, exempelvis en game over-skärm, en startmeny eller en high score-lista. Ett detaljerat diagram som beskriver förhållandet mellan spelets klasser kan ses nedan:



Varje Game-objekt i Game_State existerar oberoende av varandra, men för att föra vidare varje nödvändig information mellan spelets olika nivåer, vilka hanteras i varsitt Game-objekt, sparar Game_State undan den nödvändiga informationen i variabler och använder sedan dessa variabler när nästa Game-objekt skapas.

Alla objekt på spelplanen ärver från den abstrakta klassen Object, som i sin tur ärver från den abstrakta SFML-klassen sf::Drawable. Detta arv medför att samtliga Object på spelplanen kan skickas direkt till spelfönstrets Draw-funktion för att ritas ut, något som avsevärt förenklar ritningen av spelobjektet.

Under den abstrakta klassen Object finns två till abstrakta klasser vid namn Movable och Treasure. Movable är en abstrakt klass för samtliga objekt på spelplanen som ska ha möjlighet att röra på sig, medan Treasure är en abstrakt klass för samtliga objekt som ska kunna bli upplockade av spelaren. Samtliga objekt på spelplanen existerar oberoende av varandra och kollisionsdetektionen görs med hjälp av sf::Sprite-klassens inbyggda kollisionshantering, detta åstadkoms genom att samtliga objekt har en sf::Sprite-variabel tack vare arvet från sf::Drawable, så med detta designval fick vi enkel uppritning av objektet och kollisionshantering i ett, vilket känns som ett väldigt lämpligt designval.



Tänkbara nackdelar med designvalen är exempelvis valet att göra Object en underklass till sf::Drawable, vilket betyder att en hel del ändringar skulle behöva göras om vi i framtiden bestämmer att vi helt vill separera på de logiska spelobjekten och dess grafiska representation, men detta var en medveten uppoffring av oss för att komma åt fördelarna med Drawable, som den tidigare nämnda inbyggda kollisionshanteringen samt stödet med att skicka hela objektet till RenderWindows-klassens draw-funktion för att rita ut objektet.

3 Detaljbeskrivning av klassen Player

Klassen Players syfte är att representera spelarobjektet på spelplanen. Skillnaden på spelarens karaktär och andra objekt är att det endast får finnas ett spelarobjekt per spelplan och att det styrs med tangentbordet. Player-klassen ärver från Movable-klassen, vilken är en abstrakt klass för samtliga objekt på spelplanen som har möjlighet att påverka varandra vid kollision. Movable-klassen ärver i sin tur från klassen Object.

3.1 Playerklassens arv

Från Movable ärver Player följande funktioner:

- *virtual void attack(movable&)* - Spelaren attackerar det Movable-objekt som skickas in som parameter. Skadan som görs är ett slumpmässigt valt värde mellan *min_attack* och *max_attack* + *equipment_bonus* + (spelarens level-1)*5 vilket innebär att spelarobjektets skada ökar ju högre level det är.
- *void decreaseHp(int)* - Minskar *current_health* med medföljande integer.
- *void heal()* - Sätter *current_health* till *max_health*.
- *bool active()* const - Kollar att *current_health* är mer än noll och returnerar true om så är fallet.
- *void move(int, int)* - Förflyttar objektet genom att ange hur många gånger sin width objektet ska flyttas i x- respektive y-led.

Från Movable ärver Player även följande variabler:

- *int equipment_bonus* - Modifierar hur mycket skada spelaren kan ta och ge.
- *int max_health* - Spelarobjektets maximala hälsopoäng.
- *int current_health* - Spelarobjektets nuvarande hälsopoäng.
- *int min_attack* - Hur mycket skada objektet kan göra som minst.
- *int max_attack* - Hur mycket skada objektet kan göra som mest.

3.2 Playerklassens funktioner och variabler

Klassen har även ett antal egna funktioner och variabler:

public:

- *void increaseScore(int)* - Ökar spelarens *score* med inparameterns värde.
- *void increaseExperience(int)* - Ökar spelarens *experience_points* med inparameterns värde. Om spelarens erfarenhetspoäng överskrider 50+(*level**3) så ökas spelarens *level*, vilket ökar spelarens *current_health*, spelarens *max_health*, samt spelarens *level*-variabel, samtidigt som erfarenhetspoängen minskar med de antal poäng som krävdes för att öka en spelarnivå.
- *void equipTreasure(int)* - Spelarens *equipment_bonus* ökar med inparameterns värde och spelarens *sprite* ändras.
- *int getScore()* - Returnerar hur många poäng spelaren har.
- *int getExperiencePoints()* - Returnerar hur många erfarenhetspoäng spelaren har.
- *int getHealth()* - Returnerar spelarens *current_health*.
- *int getMaxHealth()* - Returnerar spelarens *max_health*.

- *int getLevel()* - Returnerar spelarens *level*.
- *void setLevel(int)* - Sätter spelarens *level* till inparameterns värde, samt spelarens *current_health* och *max_health* till 90+(*level**10).
- *void setExperience(int)* - Sätter *experience_points* till inparameterns värde.
- *void setScore(int)* - Sätter *score* till inparameterns värde.

private:

- *void changeSprite()* - Ändrar spelarobjektets sprite.
- *int experience_points* - Spelarobjektets erfarenhetspoäng.
- *int score* - Spelarobjektets poäng.
- *int level* - Spelarobjektets level, initieras alltid till 1.

4 Detaljbeskrivning av klassen Level

Klassen Level ärver ingenting från andra klasser, men används för att generera ett level-objekt baserat på en .map-fil. Klassen har följande funktioner:

public:

- *vector<Monster> getMonsters() const* - Returnerar en vektor innehållandes alla Monster-objekt.
- *vector<Wall> getWalls() const* - Returnerar en vektor innehållandes alla Wall-objekt
- *vector<Treasure*> getTreasures() const* - Returnerar en vektor innehållandes Treasure-pekare.
- *Goal getGoal() const* - Returnerar *goal*-variabeln.
- *Player getPlayer() const* - Returnerar *player*-variabeln.
- *Sprite getBg() const* - Returnerar *background*-variabeln.
- *Sprite getStatus() const* - Returnerar *player_status*-variabeln.

private:

- *void readLevel()* - en funktion som läser in nuvarande levels objekt från en fil till lämpliga variabler och vektorer.

Klassen Level har även följande privata variabler:

- *int level* - Anger vilken nivå spelaren är på.
- *sf::Texture tileset* - Texturen som används för att skapa sprites åt alla Objects förutom Potion och Equipment.
- *sf::Texture i_tiles* - Texturen som används för att skapa sprites åt Potion och Equipment.
- *vector<Monster> monsters* - En vektor som innehåller alla Monster-objekt.
- *vector<Treasure*> treasures* - En vektor som innehåller Treasure-pekare.
- *vector<Wall> walls* - En vektor som innehåller alla Wall-objekt.
- *Goal goal* - Nivåns Goal-objekt.
- *Player player* - Nivåns Player-objekt.
- *Sprite background* - Nivåns bakgrunds-sprite..
- *Sprite player_status* - Nivåns ruta för spelarobjektets information.

5 Externa filformat

Det externa filformatet är en vanlig textfil med filändelsen .map, dessa används för att ladda in spelets nivåer och varje .map-fil innehåller en separat nivå.

Filernas utformning är att de första raderna innehåller instruktioner för att skapa nivåns patrullerande monster, värdena på dessa rader är separerade med mellanslag för att ge stöd för formaterad inmatning. Efter dessa inledande specialrader kommer sedan ett par rader där varje tecken på raden motsvarar en ruta i spelets logik, och där olika tecken motsvarar olika objekt på spelplanen, exempelvis väggobjekt, spelarobjekt eller skattobjekt.