

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

Love Bäckman, lovba497@student.liu.se
Gustav P Svensson, gussv375@student.liu.se

Innehåll

1	Revisionshistorik	2
2	Detaljbeskrivning av Player	2
2.1	get_type	3
2.2	is_solid	3
2.3	get_delete_status	3
2.4	get_shape	3
2.5	simulate	4
2.6	end_simulate	4
2.7	prepare_simulate	4
2.8	handle_moving_collision	5
2.9	handle_static_collision	5
2.10	handle_end_collision	5
2.11	collision_state_cleanup	5

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Utkast	161110

2 Detaljbeskrivning av Player

Playerklassen representerar spelarkaraktären som spelaren styr med hjälp av piltangenterna.

Spelaren ärver från klassen `Gravitating_Object`, som ärver från klassen `Movable_Object`, som ärver från klassen `Simulatable_Object`, som ärver från klassen `Object`. Det vill säga, spelaren är ett simulerbart och rörligt objekt som påverkas av gravitation.

Player's konstruktor tar emot en `Vector2f` för position, en `Vector2f` för storlek, en sträng för typ, en `Texture`-pekare för textur och en float för hastighet.

Samtliga parametrar förutom hastigheten skickas vidare till `Gravitating_Object`'s konstruktor, tillsammans med ett falskt booleskt värde som talar om att spelaren är icke-solid. Hastigheten lagras i Player's `m_speed`-variabel.

Player ärver följande konstant-variabler från sina superklasser (enligt stilen **variabel** - typ av konstant - beskrivning):

`const sf::Texture *const m_texture` - textures - textur som spelaren ska ritas ut med

`const std::string m_type` - attributes - identifierar att spelaren är av typ "player"

Player introducerar följande nya konstant-variabler (enligt stilen **variabel** - typ av konstant - beskrivning)

`const float speed` - attributes - definierar spelaren bashastighet

Player ärver följande state-variabler från sina superklasser (enligt stilen **variabel** - typ av state - beskrivning):

`bool m_solid` - attributes - om spelaren är solid (alltid falsk)

`bool m_delete_status` - general - om spelaren ska deletas (alltid falsk)

`sf::RectangleShape __shape` - general - shape som används för att få ut position, storlek samt för utritning

Player introducerar följande nya state-variabler (enligt stilen **variabel** - typ av state - beskrivning):

`bool m_jumping` - general - om spelaren hoppar

`bool m_on_ground` - general - om spelaren är på marken

`sf::Clock m_slow_bird_clock` - buffs & debuffs - hur länge sedan det var spelaren kolliderade med en `Slow_Bird` (eller `Bomb_Bird`)

`std::unordered_set<const Object *const> m_slow_bird_debuffs` - buffs & debuffs - samtliga `Slow_Bird`'s (och `Bomb_Bird`'s) vars debuffs är aktiva på spelaren

`sf::Clock m_boost_bird_clock` - buffs & debuffs - hur länge sedan det var spelaren kolliderade med en `Boost_Bird`

std::unordered_set<const Object *const> m_bost_bird_buffs - buffs & debuffs - samtliga Boost_Bird's vars buffs är aktiva på spelaren

sf::Clock m_nfbb_clock - buffs & debuffs - hur länge sedan det var spelaren kolliderade med en NFBB

int m_nfbb_debuffs - buffs & debuffs - antal aktiva NFBB debuffs

bool m_quicksand_debuff - buffs & debuffs - om Quicksand debuffen är aktiv

bool m_quicksand_collision - collision - om spelaren kolliderat med ett block av typ Quicksand

Player ärver följande metoder från sina superklasser:

- `get_type`
- `is_solid`
- `get_delete_status`
- `get_shape`
- `simulate`
- `end_simulate`

2.1 `get_type`

Parametrar:

Return: *std::string*

Metoden `get_type` returnerar `m_type`-variabeln.

2.2 `is_solid`

Parametrar:

Return: *bool*

Metoden `is_solid` returnerar `m_solid`-variabeln.

2.3 `get_delete_status`

Parametrar:

Return: *bool*

Metoden `get_delete_status` returnerar `m_delete_status`-variabeln.

2.4 `get_shape`

Parametrar:

Return: *sf::RectangleShape*

Metoden `get_shape` returnerar `m_shape`-variabeln.

2.5 simulate

Parametrar: *const int total_simulations, const std::vector<const Object *const> &objects*

Return: *std::vector<Object *const>*

Metoden `simulate` utför ett objekts simuleringslogik. För `Player`-klassen innebär detta att förflytta sig samt kolla utföra kollisionsskontroller.

2.6 end_simulate

Parametrar: *const std::vector<const Object *const> &objects* Return:

Metoden `end_simulate` utför logik som skall utföras efter att ett objekt simulerat färdigt. För `Player`-klassen innebär detta att utföra en sista kollisionsskontroll samt återställa simulation state-variabler.

`Player` ärver och override:ar följande metoder från sina superklasser:

- `prepare_simulate`
- `handle_moving_collision(const Object *const object, const sf::Vector2f &steps)`
- `handle_static_collision(const Object *const object)`
- `handle_end_collision()`
- `collision_state_cleanup()`

2.7 prepare_simulate

Parametrar: *const float distance_modifier, const float gravity_constant*

Return: *int*

Metoden `prepare_simulate` förbereder spelaren för simulering genom att räkna ut vilken riktning spelaren ska röra sig åt beroende på vilka tangenter som är nedtryckta, samt hur lång distansen som spelaren ska röra sig blir efter att ha applicerat `m_speed`-variabeln och en `speed_modifier`-variabel (som räknas ut beroende på spelarens state).

Resultatet av uträkningarna lagras i `m_distance`-variabeln. Efter uträkningarna är klara anropas den override:ade metoden, `Gravitating_Object::prepare_simulate`, med samma parametrar.

`Gravitating_Object::prepare_simulate` räknar ut gravitationspåverkan och applicerar uträkningen på `m_distance`-variabeln, varefter den override:ade metoden `Movable_Object::prepare_simulate` anropas med samma parametrar.

`Movable_Object::prepare_simulate` applicerar `distance_modifier` på `m_distance` och räknar ut hur många simulationer som behöver göras med restriktionen att ett objekt inte kan förflytta sig mer än 24px (det minsta existerande objektet) åt gången. Detta för att undvika missade kollisioner om ett objekt försöker röra sig över 24px. Om objektet försöker röra sig 30px kommer alltså antal behövda simulationer vara 2.

`Movable_Object::prepare_simulate` returnerar sedan antalet behövda simulationer till `Gravitating_Object::prepare_simulate`, som returnerar vidare till `prepare_simulate` som returnerar värdet till sin anropare.

2.8 `handle_moving_collision`

Parametrar: *const Object *const object, const sf::Vector2f &steps*

Return:

Metoden `handle_moving_collision` hanterar kollisioner mellan spelaren och objekt som uppstår medan spelaren försöker förflytta sig. Metoden börjar med att anropa sina superklassers implementationer av metoden, för att sedan utföra logik specifik för `Player`-klassen. I metodkedjan för `handle_moving_collision` hanteras framförallt kollisioner mellan spelare och solida objekt, vid vilka spelarens position justeras så att denne inte kan passera igenom dem.

2.9 `handle_static_collision`

Parametrar: *const Object *const object*

Return:

Metoden `handle_moving_collision` hanterar kollisioner mellan spelaren och objekt som uppstår före och efter att spelaren har förflyttat sig. Metoden börjar med att anropa sina superklassers implementationer av metoden, för att sedan utföra logik specifik för `Player`-klassen. I metodkedjan för `handle_moving_collision` hanteras kollisioner med andra rörliga objekt, vid vilka spelarens state påverkas.

2.10 `handle_end_collision`

Parametrar:

Return:

Metoden `handle_end_collision` utför logik som ska utföras efter all annan kollisionslogik baserat på vad som har och inte har kolliderats med under de tidigare kollisionskontrollerna (collision state). Efter logiken utförs anropas superklassernas implementationer av metoden.

2.11 `collision_state_cleanup`

Parametrar:

Return:

Metoden `collision_state_cleanup` återställer collision state-variabler introducerade i `Player`-klassen och anropar superklassernas implementationer av metoden.