GAME DEVELOPMENT

Esercizio 2

SUPSI

Contenuto realizzato da:
Marino Alge

Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

Marino Alge Eng. ETH
marino.alge@supsi.ch

INPUT & CONTROL

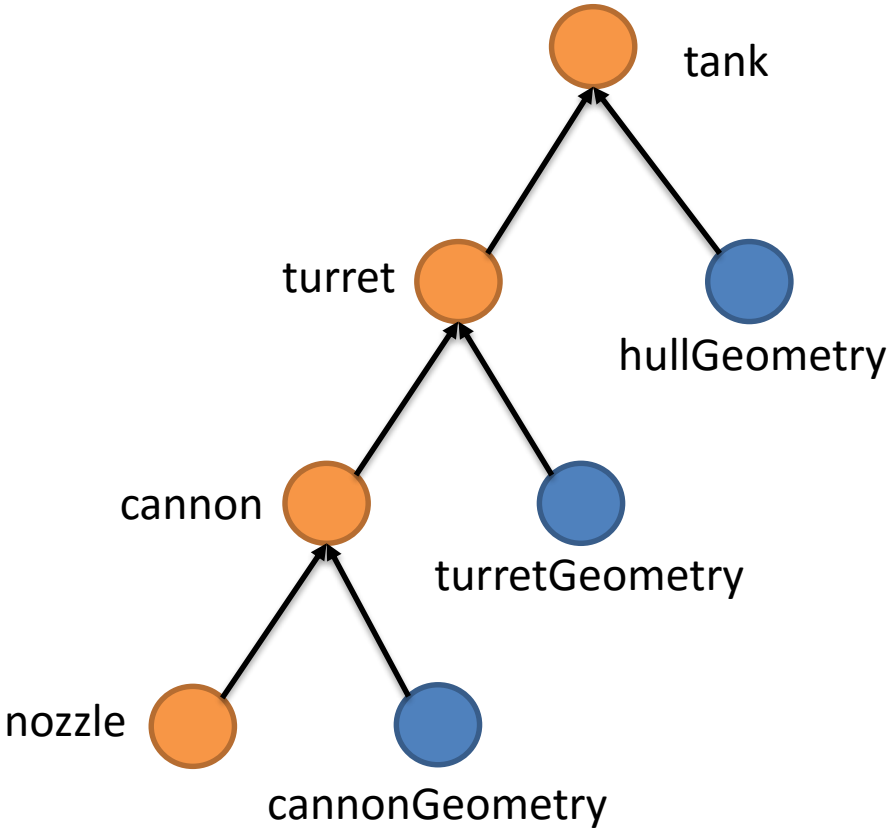
Nel **esercizio 2** aggiungeremo logica di controllo al nostro tank e la possibilità di controllarlo con mouse e tastiera.

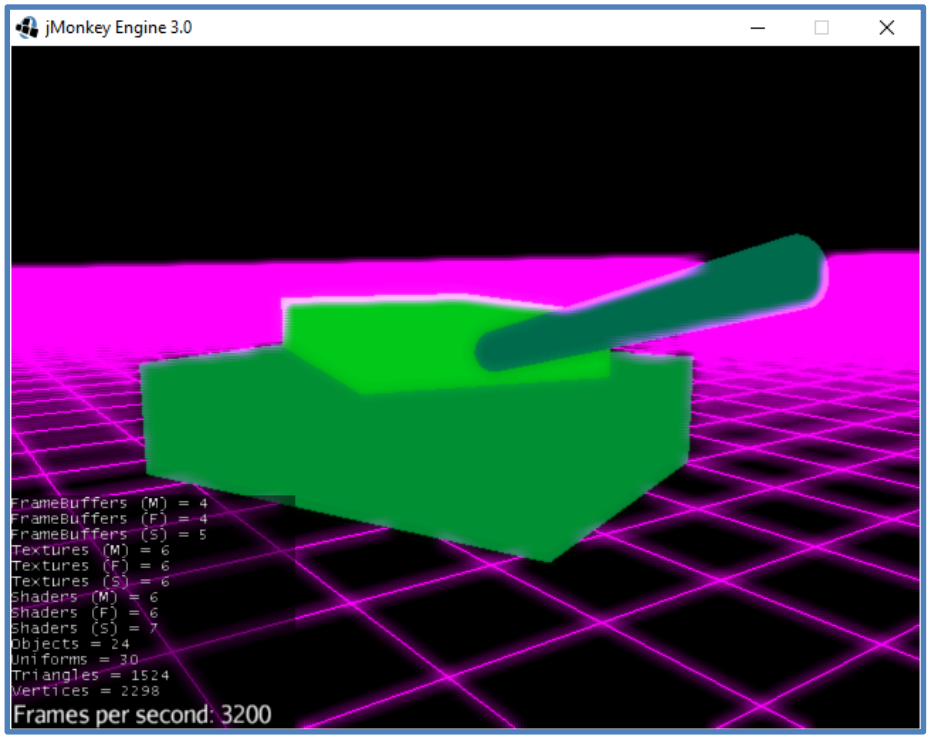
Per quanto le aggiunte siano incrementali rispetto l'esercizio 1, vi consiglio comunque di partire dal progetto «esercizio1» fornito su iCorsi2 per i seguenti motivi:

- Un po' di codice è già aggiunto per semplificarvi / accorciare il lavoro.
- Differenze d'implementazione fra la vostra versione e quella fornita potrebbero causare incompatibilità.
- Siete sicuri che eventuali errori introdotti nei precedenti esercizi non vi mordino la coda.

TANK

La struttura cinematica del tank è la seguente con i relativi SIDs.





ESERCIZIO 2.1

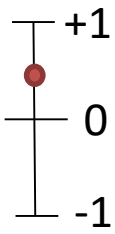
Creare la classe **`KinematicTankControl`** implementando l'interfaccia **`TankControl`** in modo che consideri le seguenti proprietà:

- **`forwardSpeed`**: velocità massima in avanti [WU/sec].
- **`backwardSpeed`**: velocità massima indietro [WU/sec].
- **`steeringSpeed`**: velocità angolare massima di curva [RAD/sec].
- **`turretSpeed`**: velocità angolare massima della torretta [RAD/sec].
- **`cannonSpeed`**: velocità angolare massima del cannone [RAD/sec].
- **`reloadTime`**: tempo di ricarica del proiettile [sec].

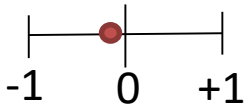
ESERCIZIO 2.1

Il **`KinematicTankControl`** deve esporre (ed implementare) la seguente interfaccia di controllo:

- **`setThrottle(float throttle)`**: fattore di velocità del tank.
  - Range di valori da -1 a 1.
  - A 1 massima velocità avanti, -1 massiva velocità indietro, 0 fermo.



- **`setSteering(float steering)`**: fattore di curva del tank.
  - Range di valori da -1 a 1.
  - A 1 massimo giro a destra, -1 a sinistra, 0 nessuno giro.
  - Un tank può girare sul posto!



- **`rotateTurret(float deltaAngle)`**: roteare la torretta.
  - La quantità di rotazione al secondo dev'essere limitata dalla velocità di rotazione massima della torretta.

ESERCIZIO 2.1

Il **`KinematicTankControl`** deve esporre (ed implementare) la seguente interfaccia di controllo:

- **`rotateCannon(float deltaAngle)`**: roteare il cannone (elevazione)
  - La quantità di rotazione al secondo dev'essere limitata dalla velocità di rotazione massima del cannone.

- **`fire()`**: sparare un proiettile
  - Permettere di sparare solo se è passato almeno il tempo di ricarica dall'ultimo colpo.
  - Non siete ancora in grado di sparare un proiettile (esercizio 2.3), per ora riservatevi l'opzione di eseguire codice in **`controlUpdate()`** nel caso si abbia sparato un colpo (**`System.out.println(«Bang!»)`**).

ESERCIZIO 2.1

Una volta implementato **`KinematicTankControl`** aggiungere una sua istanza ad ogni tank prodotto da **`TankFactory`.**

```
public class TankFactory extends SpatialFactory {
...
    @Override
    public Node newSpatial(String name) {
        ...
        // Controls
        result.addControl(new KinematicTankControl());
        ...
    }
}
```

ESERCIZIO 2.1 TIPS

Le direzioni (avanti, su, destra) di un SDR possono essere estratti dalle colonne della sua matrice di rotazione

```
Vector3f tankLeft = tank.getLocalRotation().getRotationColumn(0);
Vector3f tankUp = tank.getLocalRotation().getRotationColumn(1);
Vector3f tankAhead = tank.getLocalRotation().getRotationColumn(2);
```

Per limitare il range di una variabile è comodo utilizzare la funzione

```
public static float clamp(float value, float min, float max) {
    return value < min ? min : value > max ? max : value;
}
```

```
this.throttle = Utils.clamp(throttle, -1.0f, 1.0f);
turretDeltaAngle = Utils.clamp(turretDeltaAngle, -turretSpeed * tpf, turretSpeed * tpf);
```

Testate **`KinematicTankControl`** in **`simpleUpdate()`** della **`Main`** prima di procedere all'esercizio 2.2.

ESERCIZIO 2.2

Nel secondo esercizio daremo la possibilità al giocatore di controllare il tank.

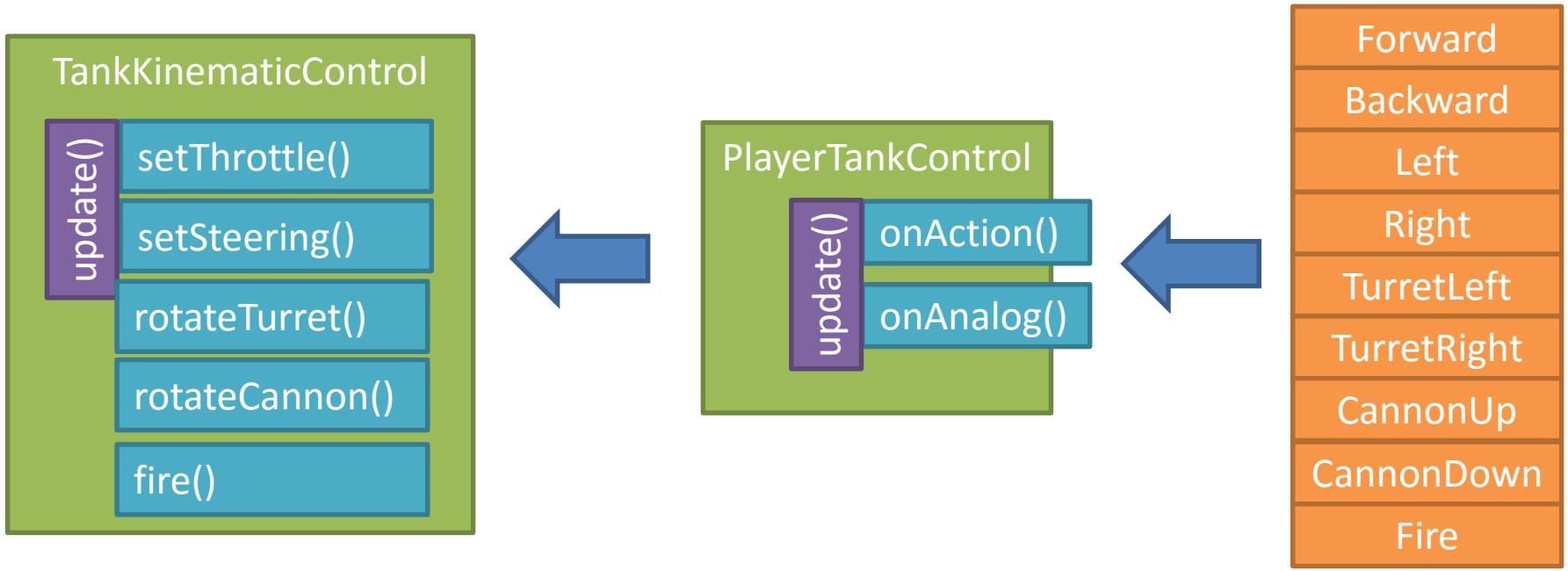
Il primo passo è definire il mapping per le seguenti azioni a mouse e tastiera nel metodo **`initInput()`** della classe **`Main`**.

- Forward
- Backward
- Left
- Right
- TurretLeft
- TurretRight
- CannonUp
- CannonDown
- Fire

ESERCIZIO 2.2

Secondo passo, creare la classe **`PlayerTankControl`** e metterla in ascolto dei eventi action & analog (implementare **`ActionListener`** & **`AnalogListener`**).

Basare **`PlayerTankControl`** su **`KinematicTankControl`** per tradurre le azioni del giocatore in movimenti del tank.



ESERCIZIO 2.2

Per finire, aggiungere una istanza di **`TankPlayerControl`** ad ogni tank prodotto da **`TankFactory.`**

```
public class TankFactory extends SpatialFactory {
...
    @Override
    public Node newSpatial(String name) {
        ...
        // Controls
        result.addControl(new KinematicTankControl());
        result.addControl(new PlayerTankControl());
        ...
    }
}
```

e metterla in listening dei eventi di input nel metodo **`initInput()`** della classe **`Main`**

```
inputManager.addListener(playerTankControl, actions);
```

Tips: Per ottenere un control di un spatial si può usare

```
Spatial tank = rootNode.getChild("tank");
PlayerTankControl playerTankControl = tank != null ? tank.getControl(PlayerTankControl.class) : null;
```

ESERCIZIO 2.3

Usare le classi (già implementate) **`SpawnControl`** e **`ProjectileFactory`** per sparare un proiettile all'evento d'input Fire dal nodo «nozzle» del tank.

```
public class SpawnControl extends AbstractControl implements Cloneable {
...
    public SpatialFactory getSpatialFactory() {...}
    public void setSpatialFactory(SpatialFactory spatialFactory) {...}
    public void spawn(String name) {...}
}
```

Setta la **`SpatialFactory`** da utilizzare per creare un nuovo spatial

Crea un nuovo spatial di nome «name» e lo aggiunge alla root del scene-graph, nella posizione del spatial a cui appartiene il control.

ESERCIZIO 2.4 (OPZIONALE)

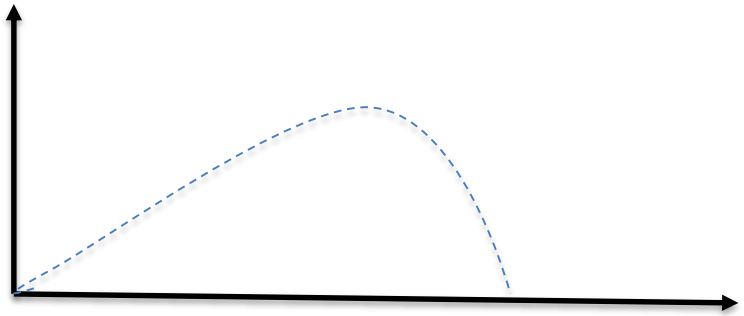
Il proiettile è controllato da **`KinematicProjectileControl`** che lo fa semplicemente procedere in linea retta per un determinato range per poi rimuoverlo dalla scena.

```
public class KinematicProjectileControl extends AbstractControl implements Cloneable {

    // Default
    private static final float DEFAULT_SPEED = 128.0f;
    private static final float DEFAULT_RANGE = 256.0f;
    // Properties
    private float speed = DEFAULT_SPEED;
    private float range = DEFAULT_RANGE;
    // State
    private float ranDistance = 0.0f;

    @Override
    protected void controlUpdate(float tpf) {...}

    @Override
    protected void controlRender(RenderManager renderManager, ViewPort viewPort) {...}

    public float getSpeed() {...}

    public void setSpeed(float speed) {...}

    public float getRange() {...}

    public void setRange(float range) {...}
}
```

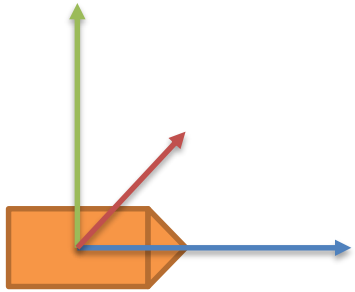
Esercizio 2.4 (Opzionale)

Creare un controllo avanzato **`AdvancedProjectileControl`** che faccia procedere il proiettile in un moto balistico per poi essere rimosso quando tocca terra (y == 0).

Le proprietà da gestire nel controllo sono:



- Velocità iniziale.
- Resistenza dell'aria.
- Gravità.



Il proiettile dovrebbe puntare sempre nella direzione di moto.

THE END

Introduzione - Transformazioni - SceneGraph

SUPSI

Contenuto realizzato da:
Marino Alge

Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

Marino Alge Eng. ETH
marino.alge@supsi.ch