U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Computer Graphics
MIEIC

Project – Supplies

State machine and controls

Teresa Matos
7-8 May 2020

Launch of supplies

Supplies are launched from the vehicle’s current position.

This launch (or fall) represents a linear animation that:

- **Starts** in the vehicle’s current position
- Has a **Y- direction** (0,-1,0)
- **Ends** when Y is null (Y=0)
- **Takes 3 seconds to reach end position**

Steps for supply creation and launch

We need to create the ***MySupply*** class and implement a state machine to control the update() and display() functions

1. Create ***MySupply*** class and its **default content and state**
2. Create *update/display* functions, adapted to **current state**
3. Create *drop/reset* functions and **apply state changes**
4. Add **supply functionalities** in ***MyScene***

1 MySupply class – Initial content

A supply is represented by geometries of your choice (e.g., ***MyUnitCubeQuad***), materials and textures.

To **implement the launch behavior**, we need:

- **State** – changes when *drop()* is called, and when position hits Y=0
- **Position** – starts at origin, changes when *drop()* is called
- **Speed** – calculated when *drop() is called*

2 Create update and display functions

The content of these functions is **dependent on the supply’s current state**

We can use **conditional statements (if/else, switch)** to perform the actions required for each state

```
update(t){
  if state == FALLING
    //Recalculate position according
    //to elapsed time
}
```

```
display(){
  if state == FALLING
    //translate to position and
    //display falling appearance
  else if state == LANDED
    //translate to position and
    //display landed appearance
}
```

2 Update Position while Falling

In the *update()* function, when supply is falling, we recalculate its position

To ensure that the **animation occurs in 3 seconds**, we need to update our state according to the elapsed time between frames

```
deltaTime = (currentTime - previousTime) / 1000;
```

To obtain deltaTime in seconds

2 Update Position while Falling

Using the **elapsed time**, we calculate the **distance** that the supply travels between previous and current frame

```
deltaDistance = deltaTime * speed;
```

Calculated in drop() function

The supply's position in Y is decremented by **delta distance;** when it reaches 0, the supply's state is changed to **LANDED**

2 Display supply according to state

When the supply is **inactive**, nothing is displayed. Otherwise, we display its geometry at the **defined position, with different appearances**.

Auxiliar display functions may be used to apply these changes:

```
display(){
  if state == FALLING
    displayFalling()
  else if state == LANDED
    displayLanded()
}
```

3 Drop function – initialize fall animation

The *drop()* function receives a *dropPosition*, **relative to the vehicle’s current position**

In this function we must:

- **Position** the supply at the received point
- Calculate the fall’s speed
- **Change supply’s state** to FALLING

```
drop(dropPosition){
  position = dropPosition
  speed = distance/fallTime;
  state = SupplyStates.FALLING;
}
```

3 Reset function – return to inactive state

The supplies also have a *reset()* function, which brings back the state defined in the constructor.

```
reset(){
  position = [0,0,0]
  speed = 0;
  state = SupplyStates.INACTIVE;
}
```

4 Supply Functionalities in MyScene

In ***MyScene***, we initialize 5 supply objects, and add the following changes.

- In the ***display()*** function, display each supply object
- In the ***update()*** function:
    - call the *update()* function for each supply
    - update *nSuppliesDelivered* variable according to supplies' state
- In the ***checkKeys()*** function:
    - call *drop()* function of an inactive supply when 'L' is pressed
    - call *reset()* function for each supply