Snake

Technical documantation

By: Sebastian Näslund

The game first initializes the game´s boundaries, speed, and a scale which is the number of pixels in which the objects should be rendered. Then it creates a window with the size of the boundaries and a renderer.

The game works by Instances of Snake objects and a food object, rendered on the screen. On construction these Snake objects and the food object are passed the data for scale and boundaries, and also a pointer to the renderer. Their colors are set and each snake get a reference to the other, and a number representing wich player it belongs to (in the case of 2 players), along with a starting lenght.

We then enter the gameloop, where on each iteration through the loop it checks for input from the user, then if the snakes are alive(bool); each snake is moved one step in it´s current direction, updates the renderer, followed by a delay determined by the speed variable. If the snakes are not alive, the player can restart the game by pressing spacebar. Movement is done by calling method move() on the Snake objects.

The Snake class is made up of a DoublyLinkedList representing it´s body, The List contain objects of a class called BodyPart, derived from class Node, representing each body part of the snake. BodyPart objects are stored in the DoublyLinkedList as Node pointers(their parent class), when methods are called to do operations on the BodyParts, they are dynamically cast into such from Nodes.

On construction the Snake calls the grow() method with the starting lenght as argument.

Page 2 : Class diagram

Page 3 : Class explanations

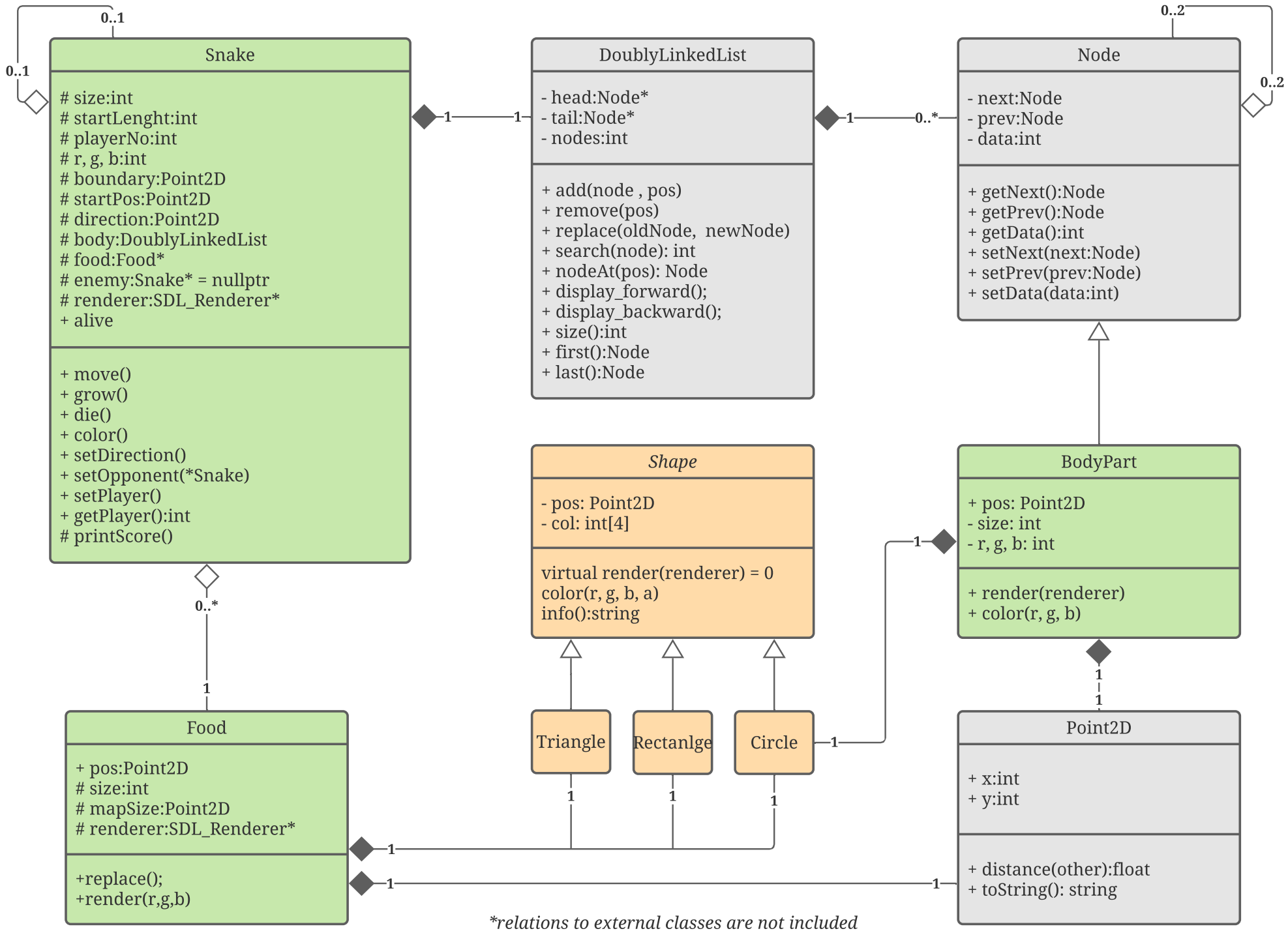
method grow(int amount). First it declares and sets two pointers, one to a new BodyPart called 'head', and the other one (called 'oldHead') is set to point at the BodyPart that´s currently at the front of the list. if the size of the list is 0(empty), meaning it´s the first time the method is being called, it sets newHeads position to the snakes startingPosition which defaults to 0,0 if not passed at construction. If the list is not empty however, it will set newHeads position to be equal to oldHead + set direction multiplied with the scale, this means it will be one step in that direction from oldHead. Next, checks are done to see if this position is within the boundaries, if it´s not, its coordinates is set to be on the inside of the boundary opposite to that of which it transgressed, so to speek.
newHead is then added to the front of the list.
Next up is collision checks, this is done by comparison of the Point2D positions, First newHeads position is compared to the position of the food, if it´s a match the method grow() is called again, then method printScore() is called. Next in the case where variable enemy Snake* is not nullptr, which means it´s 2 players. Comparison is made between newHeads position and the first BodyPart in the enemys body(its head), if they match, both snakes current length is compared, the shortest snake's die() method is then called, if they have the same lenght the direction for each snake is set to a direction corresponding to a left turn.
next it checks for collision with the enemy snakes body, by iteration, then iterating through its own body, to check for collision with itself or with food. if it´s colliding with the enemy or itself, method die() is called, if it´s colliding with food, the foods replace() method is called.
Now when the collision checks are done, if the bool alive is still set to true, newHead is rendered by calling its method render(SDL_renderer* renderer).
Lasly if amount >1 the method recursevly calls itself again with amount decremented by one.

method move(). When called, it starts by getting the last Node in the list, casting it to BodyPart, setting its color to black, and calling its render() method,thus "clearing" it from the renderer.
It then removes it from the list. and lasly it calls the grow() method to add a new bodyPart in the front of the list, thus giving it the illusion of movement.

method setDirection(int dir).
sets the direction to dir if dir is not the opposite to current direction.

method die(). prints a messege in the console, iterates through the body, rendering all bodyParts white, sets bool alive = false.

Snake class diagram*



*relations to external classes are not included

Snake

The snake class contains the bulk of the games logic, mainly methods operating on the BodyPart objects contained within the body(DoublyLinkedList).

Interesting contents:
A pointer to an enemy snake.
A pointer to a food object
A DoblyLinkedList.
Bool alive, used in gameloop.

grow()
1. creates a new BodyPart object with the coordinates from the first bodypart in the body + the direction.

2. Collision. Checks if the new BodyPart´s position corresponds to the position of any other object on the map. If collision with self/other snake, it calls the method die().

3. if still alive after collision checks, it adds the new BodyPart to the body.

move()
Moves the snake one step forward in the set direction, by removing last BodyPart, and calling method grow().

setDirection()
sets the Point2D direction coordinates x and y, to be -1,0 or +1

die()
Sets bool alive to false, this is used to control the games state.
renders all nodes in the body white.

DoublyLinkedList

The doublyLinkedList class is a container for instances of the node class, serving as a list. with methods operating on the contained nodes. i.e add/remove/search.

In the scope of the game it is used to represent the snakes body, and thus containing instances of the class BodyPart.

Node

The Node Class is a polymorphic class used in the DoublyLinkedList. It has references to the nodes before and after it, if any.

In and of itself it´s pretty useless, but by making it polymporphic it can serve as a base class for derived classes that can be used as nodes in the DoublyLinkedList, and can be accessed by dynamic casting.

Point2D

a class used to represent coordiantes in the game

Shape

Abstract class.
A visual representation on the game map

Food

As the name suggests, this is a class representing food in the game

position Point2D
a position corresponding to somewhere on the game map.

replace()
this method is called whenever a snake collides with it, it simply changes the foods position to a random location on the map.
all collision checks are done in the Snake class

Render()
Visual representation in the game using shapes

BodyPart

BodyPart is derived from the Node class. It represents a bodypart/section of the snakes body on the game map
Accessed in Snake by dynamic casting from Node.

position Point2D
a position corresponding to somewhere on the game map.
used for visual representation and collision detection.

Render()
Visual representation in the game

Triangle

derived from Shape class.
draws a triangle on the renderer

Rectangle

derived from Shape class.
draws a rectangle on the renderer

Circle

derived from Shape class.
draws a circle on the renderer