Pacman Design

By Tjeerd Bakker s2097966 and Soenke van Loh s2270862

The game will consist of a few main structures

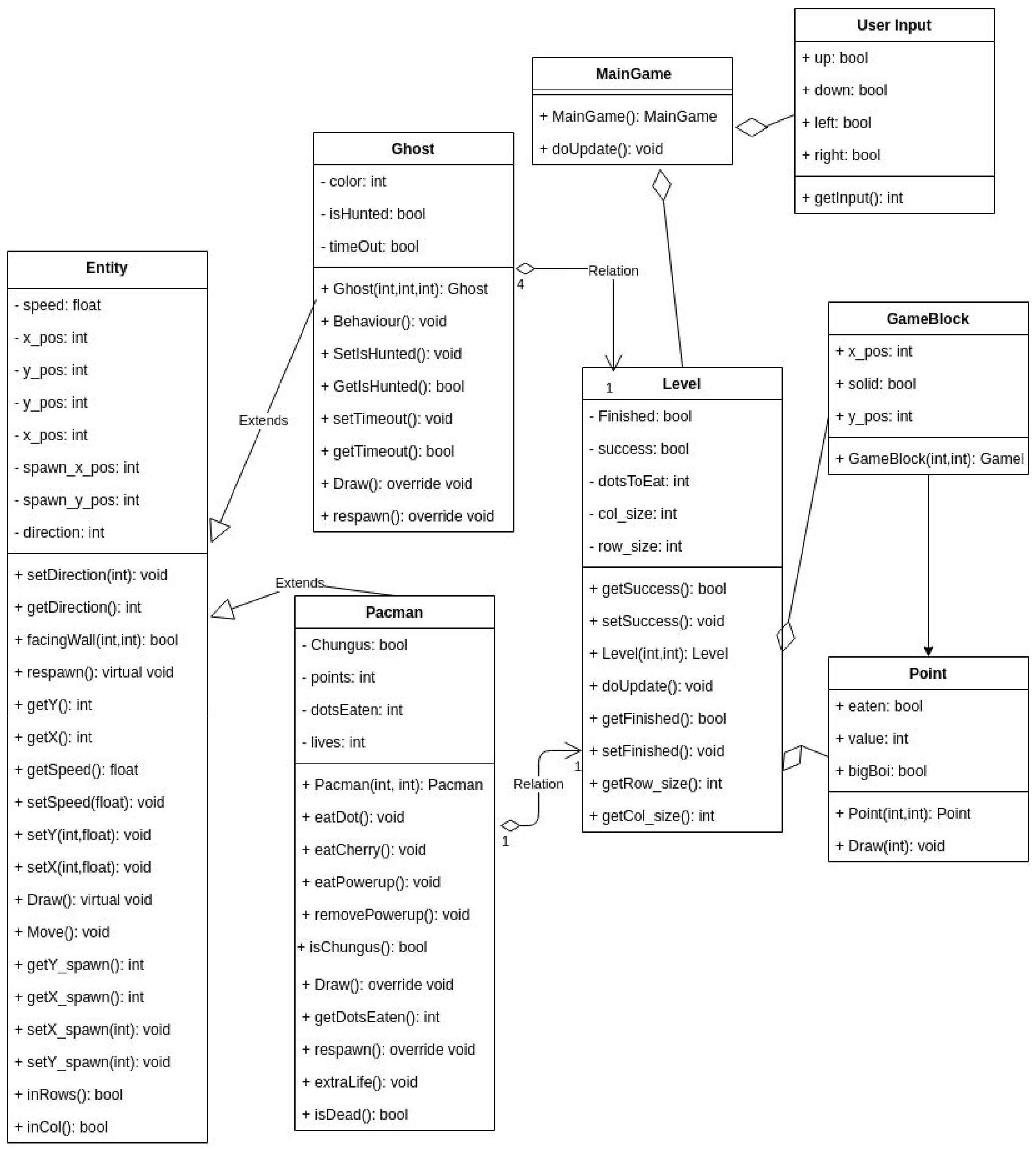
- MainGame
    - Handles user input
    - Handles timing of events
    - Holds the level
- Level
    - Holds pacman, ghosts, blocks and points
    - Controls the gameflow (Checks if pacman collides with a ghost, checks if pacman is powered up, etc)

Furthermore there will be a few objects

- Entity
    - Pacman
    - Ghost
- GameBlock
    - Pointblock (extends GameBlock)

The following considerations were taken into account:

- To allow for multiple levels, there will be a single main game handler that only controls user input, timing information (how often to update an entity) and the level object
- The position, direction, speed and other state for each entity is contained within that entity
- The Point class extends the gameblock because it uses the same position information. The PowerUp and fruits are not a separate class but rather a parameter of the point class since it can also be eaten and also has a certain score. The texture to be drawn is dependent on the score of the point since there are no 2 items that have the same score.
- The classes were aimed to be extensive but as knowledge about for example event handling with open gl and user input is not yet really established those technicalities are subject to change.
- It could happen that during the programming of the game certain functionalities contained in a single function, for example move, will be split into sub functions. This can happen if there are unexpected increases in complexity.



Inheritance Design

All classes inherit the base Package class which holds all set and get functionalities.

All Packages call the Package constructor and additionally set their own costs

using a virtual func which is overridden for every different package type.

Certain Packages inherit special package types as for example the

OvernightDelivery has to inherit TwoDayDelivery as we need the cost of that

package type as well and we have to call its specific override of calculateCost.

In case this happens the base class is indirectly inherited through that class.

PrintCosts takes the address of any instance of package class and because of the virtual

overrides always gets the according packet types costs.

QA:

1. In order to reduce code redundancy as the different package types pretty much only differ in cost
2. protected can be accessed by classes who inherit the class. Public is accessible by everyone and private only by the class itself.
3. There can be conflicting definitions in the multiple classes one inherits.
4. A computer is composed of different components as cpu, memory or mainboard. There are different types of Pc's as for example a
   gaming Pc which has everything a common Pc has and an additional graphics card. Thereby it can inherit everything from the PC and just add the graphics card.
5. At the end of the scope which determines the object's lifetime. this assumes normal stack allocation otherwise it is called with the delete keyword.
6. When we use polymorphism and assign an inherited object to a base class memory leaks can occur. When we use a virtual destructor the object will definitely be removed.
   This behaviour is very similar to how virtual is used in functions as well.