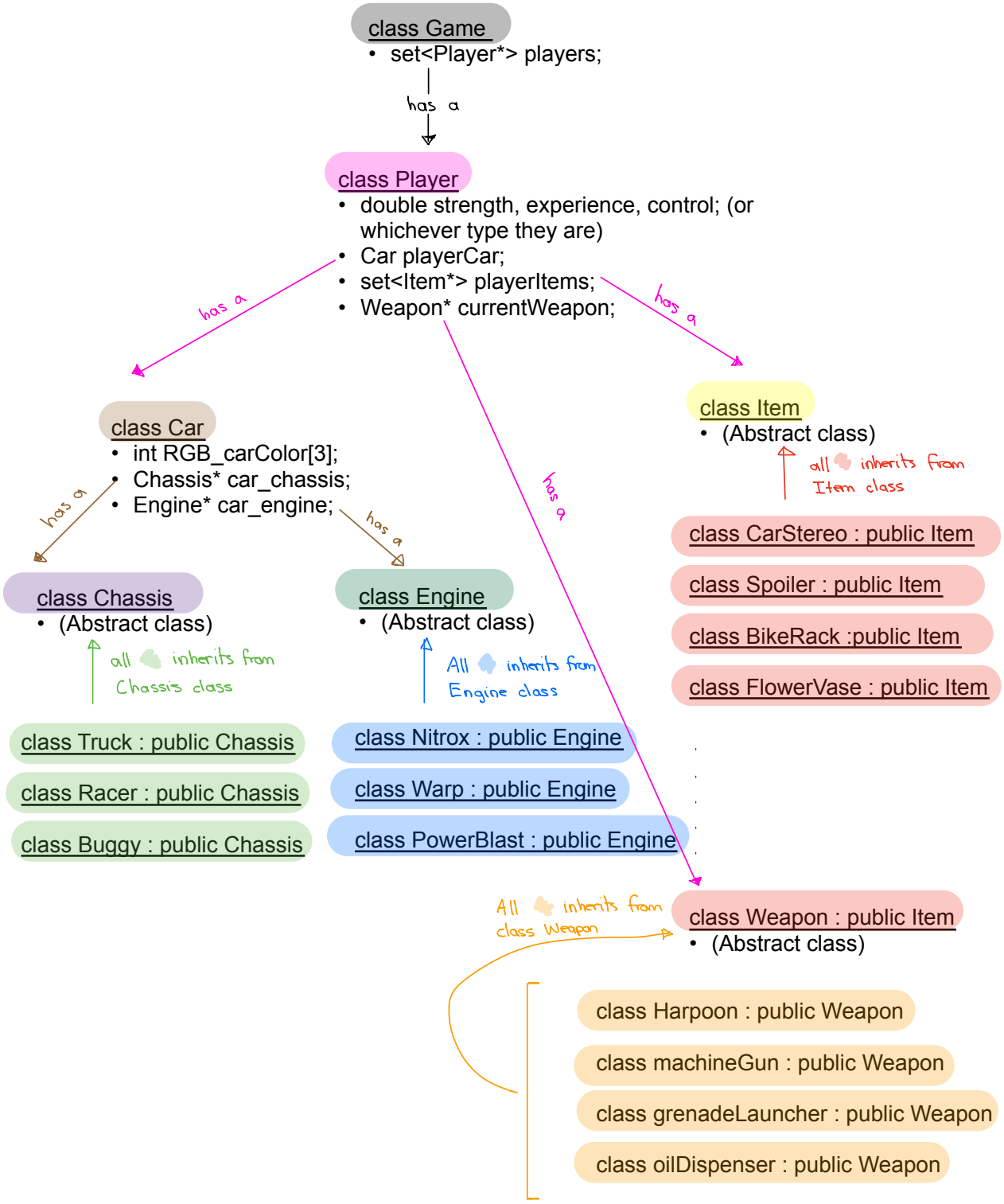
class Game
• set<Player*> players;
has a
class Player
• double strength, experience, control; (or whichever type they are)
• Car playerCar;
• set<Item*> playerItems;
• Weapon* currentWeapon;
has a
has a
has a
has a
class Car
• int RGB_carColor[3];
• Chassis* car_chassis;
• Engine* car_engine;
class Item
• (Abstract class)
all inherits from Item class
class CarStereo : public Item
class Spoiler : public Item
class BikeRack :public Item
class FlowerVase : public Item
class Chassis
• (Abstract class)
all inherits from Chassis class
class Engine
• (Abstract class)
All inherits from Engine class
class Truck : public Chassis
class Racer : public Chassis
class Buggy : public Chassis
class Nitrox : public Engine
class Warp : public Engine
class PowerBlast : public Engine
All inherits from class Weapon
class Weapon : public Item
• (Abstract class)
class Harpoon : public Weapon
class machineGun : public Weapon
class grenadeLauncher : public Weapon
class oilDispenser : public Weapon

We first want a Game class that has a set of Player objects which holds the Players that are currently playing and allows easy insertion and removal (both $\theta(logn)$ operations). The only aspect of the game specified to us so far is that the game should support an arbitrary number of players playing at a given time, so our Game class encapsulates the extent of the game requirements thus far. The Game class “has a” Player class as it stores Player pointers, and by the “new” instantiation assumed, will indirectly access Player objects as well, making it still more or less a “has a” relationship.

The Player class should store characteristics of the Player, such as strength, experience, and control (which can be assumed to be able to be stored as numeric values) as member variables—presumably as doubles or ints. We know that the player has a car (as it’s a racing game) so we give Player a Car object called playerCar. We also know that the player can accumulate an arbitrary number of Items so we give the Player a set of Item pointers (as Item will be an abstract class) called playerItems to store all the Items the Player has. Because we would also like to know what the Player’s current Weapon is (which will inherit from class Item and is an abstract class itself) we give the player a Weapon pointer currentWeapon because the Weapon class is an abstract class with different possible weapon types. This would also make it so that the Player can only use/equip one weapon at a time; we may need to first confirm with the playerItems set that the currentWeapon is contained within the set. Thus the Player class will have a “has a” relationship (directly or indirectly) with class Car, class Item, and Class Weapon.

Class Car is the Car object of the Player, so it should have a color, a Chassis, and an Engine. We first give the car its color by doing an int array[3] to store all red, green, and blue values to make whatever color desired. If Player wanted to set the Car color, Player can call a public function in class Car::setColor(int r, int g, int b) {...} or something similar accordingly. We give the Car a Chassis pointer called car_chassis to effectively assign the chassis of the car; class Chassis is an abstract class so we would want to use a pointer here. Similarly we would want to give the car an engine so we give Car a member of type Engine pointer called car_engine as Engine class is abstract.

Class Chassis is an abstract class with member variables/member functions (mostly virtual) common among all chassis. We want to keep Chassis class abstract as chassis as of now are either Truck, Racer, or Buggy, and not just a “Chassis”. This way if we wanted to just get what a given car’s chassis does for speed, then we can iterate through the chassis instead of trying to figure out what type of chassis it is first (aka the benefit of abstract class design).

Class Truck, class Racer, and class Buggy all publicly inherit from class Chassis as they have all the features of a Chassis but with additional or unique features to each type of chassis. It is publicly inherited because we would still want to be able to use their functions as well as all of Chassis’ functions in the main Game class.

Class Engine is an abstract class with member variables/member functions (mostly virtual) common among all engines. We want to keep Engine class abstract for the same reason we want the Chassis class to be abstract: as there is no car with just an “engine” as it must be one of the 3 existing ones as of now and it would be easier to later iterate through the Player car’s engines if need be instead of trying to figure out what type of engine it is.

Class Nitrox, class Warp, and class PowerBlast all publicly inherit from class Engine as they have all the features of an Engine but with additional or unique features to each type of Engine. It is publicly inherited because we would still want to be able to use their functions as well as all of Engine’s functions in the main Game class.

Class Item is an abstract class with member variables/member functions (mostly virtual) common among all Items. We want to keep Items class abstract for the same reason we want the Chassis and Engine classes to be abstract: as there is no car with a “base Item” as it must be one of the 5 existing items as of now or a Weapon and it would be easier to later iterate through the Player Items if need be instead of trying to figure out what type of Item it is. It would also not be able to store different types of unconnected classes in one singular set in the Player class, so an abstract class Item is best for this scenario.

Class CarStereo, class Spoiler, class BikeRack, class FlowerVase all publicly inherit from class Item as they have all the features of an Item but with unique/additional features for each of these respective “Item type” classes. It is publicly inherited because we would still want to be able to use their functions as well as all of Item’s functions in the main Game class. Class Weapon also publicly inherits from class Item for the same reasons.

Class Weapon is an abstract class is an abstract class with member variables/member functions (mostly virtual) common among all Weapon “types”. We want to keep the Weapons class abstract for the same reason we want the Chassis/Engine/Item classes to be abstract: as there is no Item that is just a “Weapon” as it must be one of the 4 existing Weapons (Harpoon, machineGun, grenadeLauncher, or oilDispenser) and it would be easier to later iterate/execute through the Player currentWeapon if need be instead of trying to figure out what type of Weapon it is first. We would also not know what type to set the currentWeapon pointer if it was not “bundled up” as an abstract template of sorts, so an abstract class Weapon is best for this scenario.

We would then have class Harpoon, class machineGun, class grenadeLauncher, and class oilDispenser publicly inherit from the abstract Weapons class as they all have features of a Weapon but with unique/additional feature for each respective types of weapon classes. It is publicly inherited because we would still want to be able to use their functions as well as Weapon’s functions in the main Game class as well as in classes that these weapon-type classes do not directly inherit from.