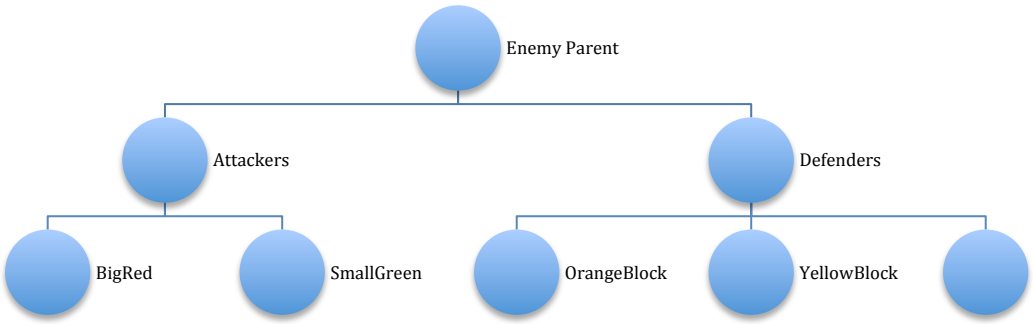
Inheritance allows us to consolidate scripts and code efficiently. Anytime you find yourself copying and pasting a function into more than one script or making multiple scripts to code similar behaviors for different game objects, you probably should be using inheritance.

Imagine that you want enemies to spawn in your game. There will be several kinds of enemies that spawn in different places and have different speeds and weapons, but they all share some attributes.

First think about all the things that objects of this type share. These shared parameters and behaviors can be defined in a parent script. All children of the parent script have all the attributes of the parent. You can give them additional features in their own subscripts:



All the enemies:

Are named
Move
Destroy the player object on contact
Take credit when kill player

Attackers: move toward player
Defenders: move back and forth

The enemy parent script is an Abstract class. This means that we don't actually attach that script to any individual game object. The script is automatically part of any child script that IS attached to an object.

For example, the BigRed script is a child of the EnemyParent script and the Attackers script. BigRed IS attached to the BigRed prefabs. Neither the EnemyParent.cs nor Attackers.cs is attached to any script.

The OrangeBlock script is a child of the EnemyParent script and the Defenders script. OrangeBlock IS attached to the OrangeBlock prefabs.

Coding EnemyParent:
create variables to be set by children
define functions to be called by children

Coding Attackers and Defenders

Functions in parent scripts can be overridden by child scripts.
We want the two subclasses to move differently so we will override the MoveMe function in the parent script with seek and patrol scripts in the Attackers and Defenders scripts

Coding subclass attackers and defenders: right now they just take credit for kills. You can customize them to do whatever you want.