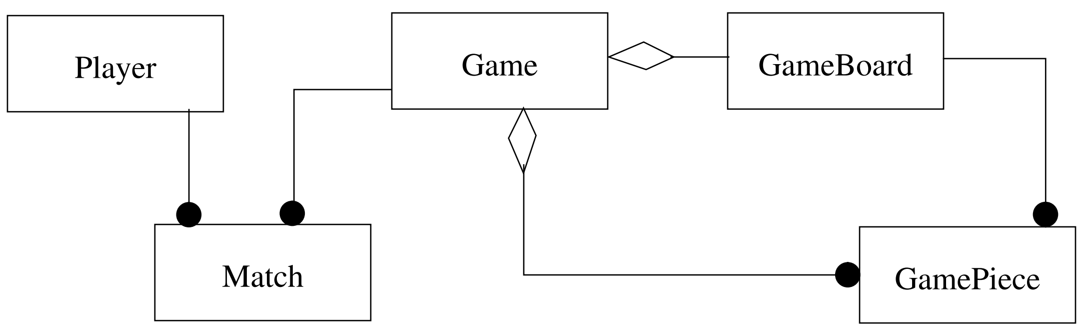
Analysis

Domain Analysis and Scope

In looking over the problem domain, it is easy to become stuck in the video game world—or even worse, in games very similar to Brickles. To attempt to gain true reuse we would like to identify a framework of classes that can be used across a variety of applications (games in this case). If we step back from the problem a bit we can consider a wide range of games. In some earlier work we identified a framework for board games such as chess and checkers. Below is an object model for that framework.



This model must be expanded to incorporate the ideas of arcade games. These games introduce the concepts of movement and obstacles to the set of concepts already contained in the framework. [Note that this is a separate framework from the framework for the graphical user interface (GUI). We will consider later how to integrate these two.]

The GamePieces in Brickles include the puck, paddle and all of the obstacles such as the walls and bricks. A match is one playing of the game such as one set of obstacles. Once these are destroyed or the number of pucks is exhausted the match is over. The GameBoard contains all of the GamePieces and obstacles.

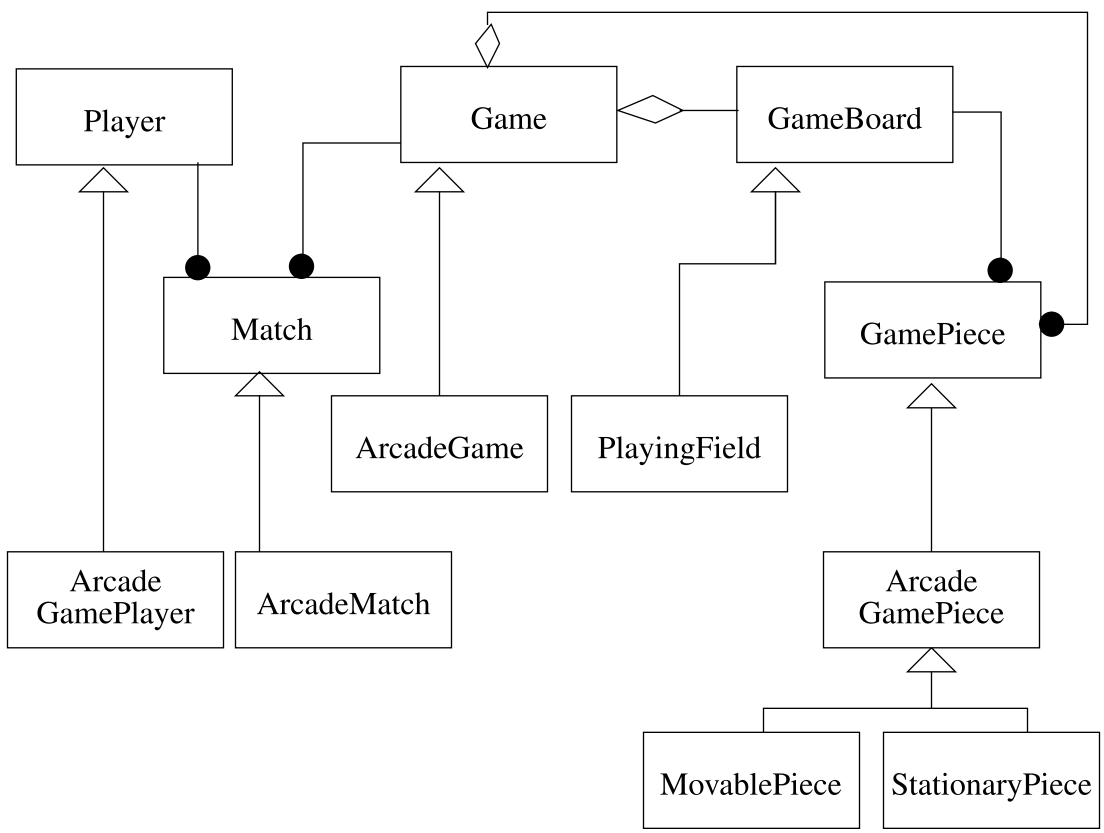
The puck has logical attributes such as *speed, direction, location*. In addition the puck must *move* appropriately, based on its current speed and direction. *None of these features are unique to a puck moving around in a game called Brickles.* In fact, they are appropriate for any object moving through a two dimensional plane.

Similarly, as we consider the ceiling, walls, floor, paddle, and the bricks, we see that each of these must respond to (and perhaps first detect) collisions with the puck. So what we really have is a special case of a very general situation that involves movable two-dimensional objects zipping around, and occasionally bumping into obstacles, at which point the obstacles respond in ways that are specific to themselves.

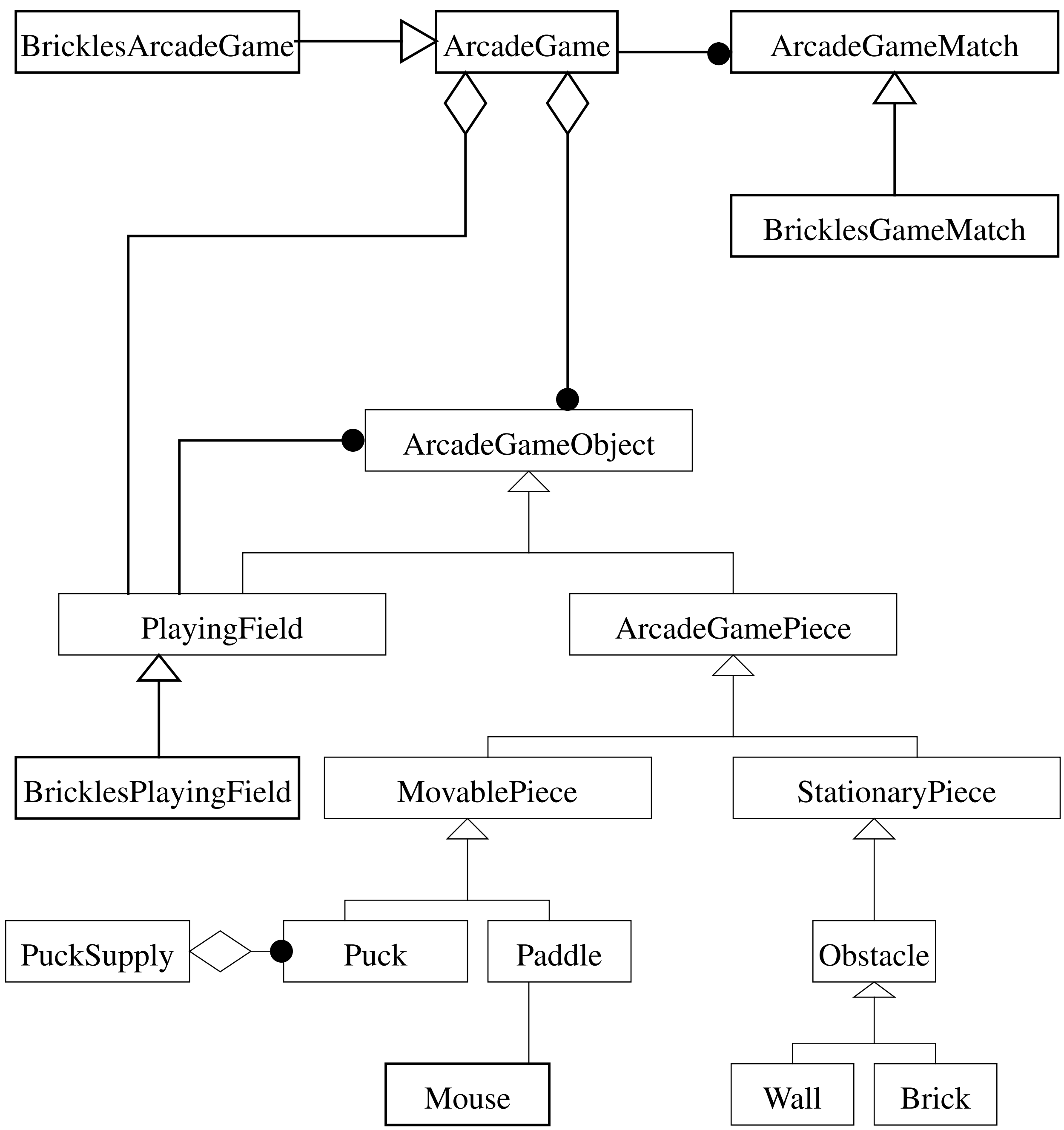
Upon making this discovery we must decide whether we really want to try to model this movable/obstacle framework. This requires doing some cost/benefit analysis to see whether the extra resources required to modify the existing design of the framework are likely to be recouped over a reasonable period of time. After due deliberation we decide to incorporate these elements into our existing framework in our design with the caveat that we will not spend too much time trying to perfect the classes at this level. We will do our best, and then expect to improve the framework when the next, similar application comes along.

We should also consider the possibility of expanding the scope even further. For example why limit the design to objects that move in just two dimensions? Again, this is a combined business/technical decision that must be made with a clear understanding of the trade-offs between expending additional resources on this project against conserving resources on (perhaps) many future projects.

The object model below shows the framework from which we intend to design the Brickles game.



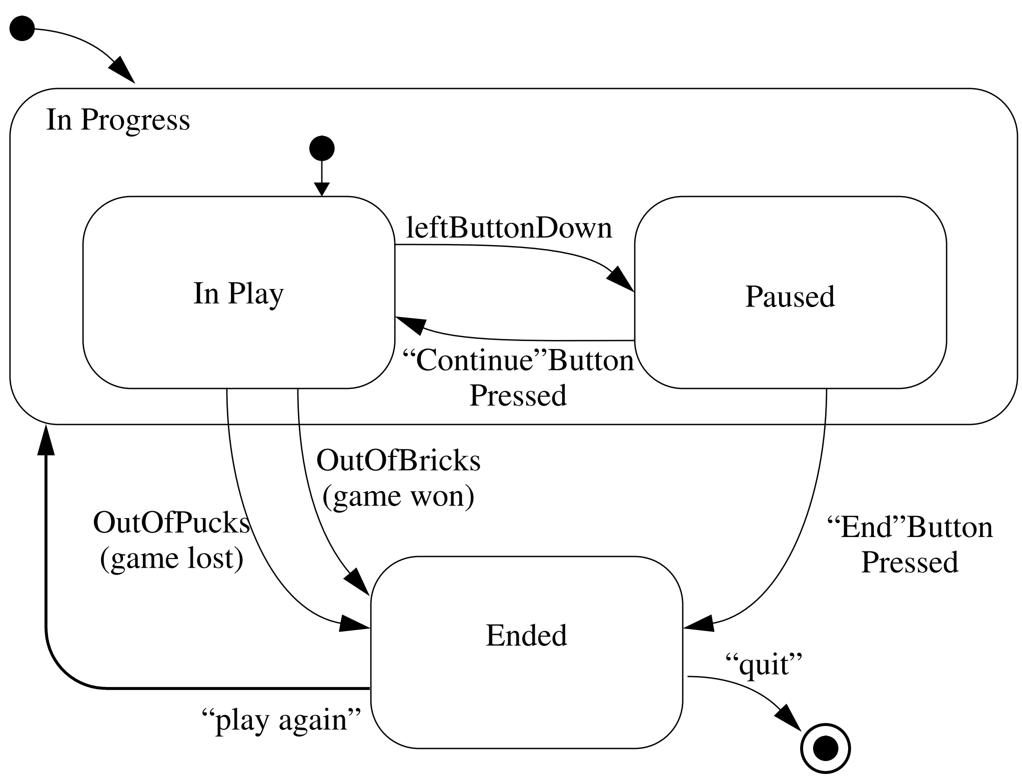
Brickles Object Model



Dynamic Models

A dynamic model defines the states of an instance of a class and the transitions between them associated with various events.

The dynamic model



describes the high level states of the *Increment 1* version of the game.

Dynamic Model (cont'd)

- MovablePiece