1. (10 points) Give the output for the following program.

```
#include <iostream>
#include <cstdlib>
#include <vector>
const int MAX = 1000;

int main() {
  std::vector<int> vec;
  for (int i = 0; i < MAX; ++i) {
    vec.push_back( rand()%100 );
  }
  vec.push_back(2);
  std::cout << vec.size() << std::endl;
  std::cout << vec.capacity() << std::endl;
}
```

****

1001

1024

****

---

2. (10 points) Give the output for the following program.

```
#include <string>
#include <vector>
#include <iostream>
class Pokemon {
public:
  Pokemon( ) : name() { std::cout << "default" << std::endl; }
  Pokemon(const std::string& n) : name(n) {
    std::cout << "conversion" << std::endl;
  }
  Pokemon(const Pokemon& p ) : name(p.name) {
    std::cout << "copy" << std::endl;
  }
private:
  std::string name;
};
int main() {
  std::vector<Pokemon> pokes;
  pokes.push_back(std::string("Larvitar"));
  pokes.push_back(Pokemon("Steelix"));
}
```

****

conversion

copy

conversion

copy

copy

****

3. (10 points) Give the output for the following program. Note the use of reserve on line 18.

```
1  #include <string>
2  #include <vector>
3  #include <iostream>
4  class Pokemon {
5  public:
6    Pokemon( ) : name() { std::cout << "default" << std::endl; }
7    Pokemon(const std::string& n) : name(n) {
8      std::cout << "conversion" << std::endl;
9    }
10   Pokemon(const Pokemon& p ) : name(p.name) {
11     std::cout << "copy" << std::endl;
12   }
13 private:
14   std::string name;
15 };
16 int main() {
17   std::vector<Pokemon> pokes;
18   pokes.reserve(2);
19   pokes.push_back(std::string("Larvitar"));
20   pokes.push_back(Pokemon("Steelix"));
21 }
```

```
**********************************************
conversion
copy
conversion
copy
**********************************************
```

4. (10 points) Convert class Clock to a GoF singleton (the one with a static class instance variable). Be sure to "Explicitly disallow the use of compiler-generated functions you do not want."

```
#include <iostream>

class Clock {
public:
  static Clock* getInstance() {
    if ( !instance ) instance = new Clock;
    return instance;
  }
  int getTicks() const { return ticks; }
  void update() { ++ticks; }
private:
  int ticks;
  static Clock* instance;
  Clock() : ticks(0) {}
  Clock(const Clock&);
  Clock&  operator=(const Clock&);
};

Clock* Clock::instance = NULL;
int main( ) {
  Clock* clock = Clock::getInstance();
  clock->update();
  std::cout << clock->getTicks() << std::endl;
}
```

5. (15 points)

   (a) Give the output for the following program.

   (b) Make the program more efficient by preferring initialization to assignment?

```
1  #include <string>
2  #include <vector>
3  #include <iostream>
4  class Pokemon {
5  public:
6    Pokemon( ) : name() { std::cout << "default" << std::endl; }
7    Pokemon(const std::string& n) : name(n) {
8      std::cout << "conversion" << std::endl;
9    }
10   Pokemon(const Pokemon& p ) : name(p.name) {
11     std::cout << "copy" << std::endl;
12   }
13   Pokemon& operator=(const Pokemon&) {
14     std::cout << "assignment" << std::endl;
15     return *this;
16   }
17 private:
18   std::string name;
19 };
20
21 class Pokedex {
22 public:
23   Pokedex(const std::string& s) : pokemon(s) { }
24 private:
25   Pokemon pokemon;
26 };
27
28 int main() {
29   Pokedex p("Steelix");
30 }
```

```
**********************************************
conversion
**********************************************
```

6. (15 points) Give the output for the following program.

```
#include <iostream>

class A {
public:
  A(const std::string& n) : name(n) {}
   const std::string& getName() const { return name; }
   void setName(const std::string& n) { name = n; }

   virtual void display() const {
     std::cout << name << std::endl;
   }
   char getIt() const { return 'A'; }
   void onlyHere() const { std::cout << "Only in A" << std::endl; }
private:
   std::string name;
};

class B : public A {
public:
  B(const std::string& name, int x) : A(name), number(x) {}
   virtual void display() const {
     std::cout << A::getName() << ", " << number << std::endl;
   }
   char getIt() const { return 'B'; }
   void onlyHere() const { std::cout << "Only in B" << std::endl; }
private:
   int number;
};

int main() {
  B b("dog", 12);
  A* x = new B("cat", 7);

   b.display();
   x->display();
   x->onlyHere();
   std::cout << b.getIt() << std::endl;
   std::cout << x->getIt() << std::endl;
}
```

```
***********************************************
dog, 12
cat, 7
Only in A
B
A
***********************************************
```

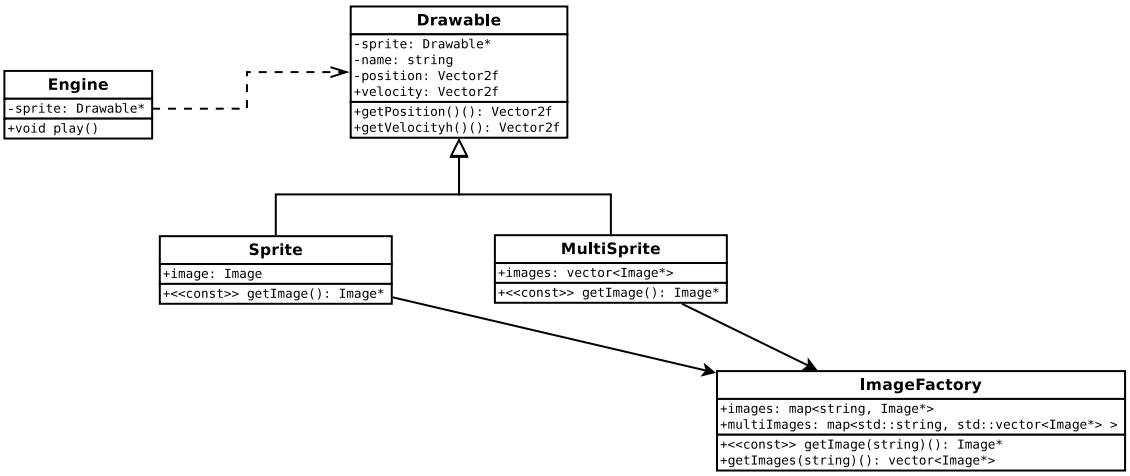

Figure 1: **Partial UML class diagram for the Game Framework**.

7. (10 points) Figure 1 illstrates a partial class diagram for selected classes in the game framework that we are using for project 3 and subsequent projects. Give short answers for the following questions:

   (a) What two functions, listed in the class diagram, are inherited by Sprite and MultiSprite?
   getPosition and getVelocity

   (b) What two important *pure virtual* functions are not listed but should be included in Drawable?
   draw and update

   (c) Write a declaration for a *polymorphic vector* in class Engine that can contain either a Sprite or a MultiSprite.

   ```
   std::vector<Drawable*> sprites;
   ```

   (d) Notice that Sprite contains a data attribute, image, a pointer to Image. Why does a MultiSprite need a vector of pointers to Image?
   A Sprite only has a single frame; however, a MultiSprite has many frames. Thus, a single instance of Image* can describe a Sprite, but a vector of Image* is needed for each frame in a multi-frame sprite.

   (e) When Engine wants to create a sprite, it uses something like:
   sprite = new Sprite(”star”);
   How is the string ”star” used so that a Sprite and an ImageFactory know the name of the file that contains the image for Sprite, the initial position and velocity of a Sprite?
   The string, “star”, is the name of the sprite. All of the information about a sprite is listed in an XML file and all of this information is contained in an XML record whose name is the same as the string, name, of a sprite.

8. (20 points) The program below uses SDL to draw a rectangle. Modify the program in the following way. Write a class, Rectangle, that contains a function draw, and other relevant data and operations to draw a rectangle. Then instantiate 6 rectangles of increasing size so that they appear as shown in Figure 2 (approximately). You can add a loop to main (below), and write class Rectangle on the following page.

```
#include <iostream>
#include <string>
#include <SDL2/SDL.h>
const int WIDTH = 640;
const int HEIGHT = 440;
const std::string TITLE = "Drawing a Rectangle";

int main (int , char*[]) {
  if ( SDL_Init(SDL_INIT_VIDEO) != 0 ) {
    return EXIT_FAILURE;
  }
  SDL_Window* window = SDL_CreateWindow(
      TITLE.c_str(), SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
      WIDTH, HEIGHT, SDL_WINDOW_SHOWN
  );
  SDL_Renderer* renderer = SDL_CreateRenderer(
    window, -1, SDL_RENDERER_ACCELERATED
  );
  SDL_SetRenderDrawColor( renderer, 255, 255, 255, 255 );
  SDL_RenderClear( renderer );

  SDL_Rect r = {150, 150, 250, 150};
  SDL_SetRenderDrawColor( renderer, 255, 0, 0, 255/2 );
  SDL_RenderDrawRect( renderer, &r );
  SDL_RenderPresent(renderer);

  SDL_Event event;
  const Uint8* keystate;
  while ( true ) {
    keystate = SDL_GetKeyboardState(0);
    if (keystate[SDL_SCANCODE_ESCAPE]) { break; }
    if (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT) {
        break;
      }
    }
  }
  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();
}
```

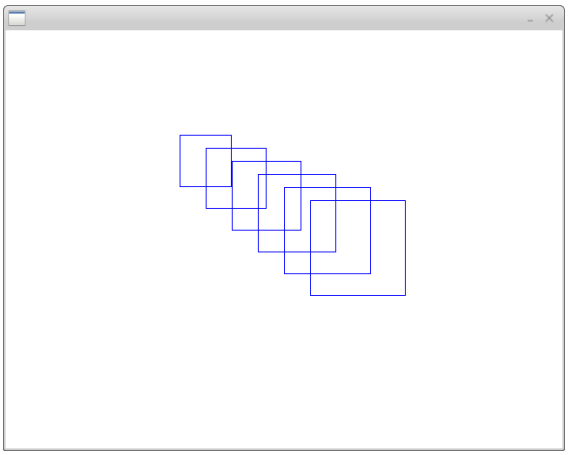

Figure 2: **Six rectangles**.

```
#include <iostream>
#include <SDL.h>

class Rectangle {
public:
  Rectangle(SDL_Renderer* rend, SDL_Color c, int x, int y, int w, int h)
    : renderer(rend), color(c), rect({x, y, w, h}) {}
  void draw() const;
  int getWidth() const  { return rect.w; }
  int getHeight() const { return rect.h; }
private:
  SDL_Renderer* renderer;
  SDL_Color color;
  SDL_Rect rect;
};
std::ostream& operator<<(std::ostream&, const Rectangle&);
```

```
#include "rectangle.h"

void Rectangle::draw() const {
  SDL_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);
  SDL_RenderDrawRect( renderer, &rect );
  SDL_RenderPresent(renderer);
}

std::ostream& operator<<(std::ostream& out, const Rectangle& rectangle) {
  return out << rectangle.getWidth() << rectangle.getHeight();
}
```

```
#include <SDL2/SDL.h>
#include "rectangle.h"
#include "frameGenerator.h"

const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const SDL_Color BLUE =  {0,0,255,255};

int main(void) {
  SDL_Renderer *renderer;
  SDL_Window *window;

  SDL_Init(SDL_INIT_VIDEO);
  SDL_CreateWindowAndRenderer(
    WINDOW_WIDTH, WINDOW_HEIGHT, 0, &window, &renderer
  );

  SDL_SetRenderDrawColor( renderer, 255, 255, 255, 255/2 );
  SDL_RenderClear(renderer);

  int w = 60, h = 60;
  int y = 120;
  for (int x = 200; x < 380; x+=30) {
    SDL_Rect rect = {x, y, w, w };
    w += 10;
    y += 15;
    Rectangle one(renderer, BLUE, rect.x, rect.y, rect.w,  rect.h);
    one.draw();
  }


  SDL_RenderPresent(renderer);
  FrameGenerator frameGen(renderer, window, WINDOW_WIDTH, WINDOW_HEIGHT,
    "malloy");
  frameGen.makeFrame();

  SDL_Event event;
  const Uint8* keystate;
  while ( true ) {
    keystate = SDL_GetKeyboardState(0);
    if (keystate[SDL_SCANCODE_ESCAPE]) { break; }
    if (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT) {
        break;
      }
    }
  }
  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();
  return EXIT_SUCCESS;
}
```