HOGESCHOOL VAN AMSTERDAM

Code Conventions

Project Game Technology

Jasper Meier, 500703267

March 2017

Glenn Hoff, Wouter van de Hauw, Michael FuentsRodriguez, Rosa Corstjens, Guus van Gend, Kevin de Leeuw, Frederick van der Meulen & Jasper Meier

Project Game Technology
Team 5

Onder begeleiding van
Alexander Mulder, Product Owner
Anders Bouwer, Scrum coach
Eric Klok, Technisch consultant

Code Conventions

Project Game Technology

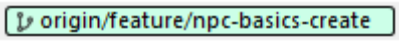
Jasper Meier, 500703267
Game Technology
Hogeschool van Amsterdam
March 2017

LANGUAGE AND NAMING

Language, severity: Moderate
All methods, variable names and comments should be written in English.

GIT

Branch creation, severity: Moderate
Branch names should be written in all lowercase. In between words a dash '-' is used instead of a space.



GIT commits, severity: Low
All commits messages should be created using the template "Added:" (added functionalities or items), "Working on:" (incomplete functionalities or items), "Removed:" (removed functionalities or items).

LIBRARIES

Library function use, severity: Moderate
When a library function is used, don't use namespaces. This creates less readable code and could cause problems when a method is overwritten by a method (with the same name) from the library.

```
MyLibrary::libraryMethod(myVariable);
```

HEADER FILES

Using header files, severity: Low
In general each class (*cpp file*) should have its own header file. There are a few small exeptions such as unit tests, and small cpp files containing only a main() function.

Self containing header files, severity: Major
Header files should be self containing i.e. compile on their own and contain the needed header files. To prevent libraries from being defined multiple times the header should also include a define guard.

The name of the define guard is a simple convention, and should follow the template: PROJECT_FILENAME_H_ as demonstrated below with the player. Define guards should only be included in header files. This prevents any header file from being included twice

```
#ifndef DIABRO_PLAYER_H_
#define DIABRO_PLAYER_H_

# include "something.h"
# include <somethingElse.h>
//the rest of the header file

#endif // DIABRO_PLAYER_H_
```

CLASSES

Class creation, severity: Critical

When multiple functionalities fit and possibly finish each other, they should be grouped within a class, however this is only when the methods could be seen as a part of each other. An example for this is the MathHelper class. Classes are also used in objects and methods with a real world relation. An example of this is a character or player class. Class names are written in camelcase, with an uppercase character in the first word of the sentence. All class names should be written in the singular form.

```
Class GameManager{
}
Class Enemy { //not Enemies
}
```

METHODS

Method names, severity: Major

When a method is created, it's name should tell what its functionality is. Names are written in camelcase with an lowercase for the first letter.

```
Void setHealth (int newHealth){
//sets health
}
int getHealth (void){
//gets health
return health;
}
```

Method length, severity: Low

There is no maximum length for methods, however don't use redundant code. Code is redundant when it is implemented more than once. In these occasions you should make another method.

//private methods

Parameters with default values, severity: Low

When a parameter has a default value i.e. a value for when the parameter is not passed to the method, this variable should be the last or second to last (in case of nullable parameters) in the total of parameters.

```
void myMethod(int myVariable, bool myDefaultValue = true, int myNullableN){
```

Nullable parameters, severity: Moderate

When a method holds certain parameters which normally could not contain the value NULL (for example int, float or boolean), i.e. nullable variables, should be passed as the last parameters of a method. To improve readability of these variables and show programmers that the variables can actually contain the NULL value the name-suffix for these variables should be N.

```
void myMethod(int myVariable, int myNullableN){
```

Retrieving nullables from the database, severity: Major

When a variable is retrieved from the database and it is nullable, create a check to catch this and prevent a nullpointer.

```
If (myNullableN != NULL){
        myVariable = myNullableN;
} else {
        //Do something else
```

```
}
```

VARIABLES

Variable creation, severity: Major

Any value, representing something (for example speed, health or position), which is used more than once should become a variable. Variables can also be created for readability when they're used once.

Variable names, severity: Low

Variable names are written in camelcase and start with an lower case letter. Camelcase implies all new words start with a capital letter (except for the first in our case).

```
Int myVariable = 0;
```

Constant variables, severity: Moderate

Where possible, the use of constant variables is mandatory. A variable is constant when it's value doesn't change in a complete run of the program. If a value is used only once, a number may be used for the constant, but if there is any doubt about the readability of this implementation, just use a variable. Constant variable names should be written in all caps, with underscores '_' instead of spaces.

```
const int MY_LENGTH = 10;
arrayVariable[MY_LENGTH]
// or if used once
arrayVariable[10];
```

Private and protected variables, severity: Low

To improve readability of the program, private and protected variables should be named with an underscore in front of their name.

```
Private:
        int _myPrivateVar = 8;
Protected:
        int _myProtectedVar = 7;
```

Parameter variables, severity: Low

When a variable is passed through as a parameter, but it has the same name as a class variable, put a small letter p in front of the variable name i.e. pMyParameter. This p stands for 'parameter'.

```
void myMethod(int pMyHealth){
        myHealth = pMyHealth;
```

IF STATEMENTS AND SWITCHES

Switch statements, severity: Major

If multiple if statements are needed on one variable, use a switch statement.

```
Switch (myInt){
Case 0: //Do something
        break;
Case 1://Do something else
        break;
default://Do something when no case is met
        break;
```