



---

---

**Project: 30%**

---

---

**Course Identification**

---

Name of program – Code:	COMPUTER SCIENCE TECHNOLOGY – VIDEO GAME PROGRAMMING – 420.BX
Course title:	<b>OBJECT-ORIENTED PROGRAMMING AND CONCEPTS II</b>
Course number:	420-JV7-AS
Group:	<b>07342</b>
Teacher's name:	Jean-François Parent
Duration:	Extended
Semester:	Fall 2021

---

---

**Student Identification**

---

Name: \_\_\_\_\_ Student number: \_\_\_\_\_  
Date: \_\_\_\_\_ Result: \_\_\_\_\_

☐ I declare that this is an original work, and that I credited all content sources of which I am not the author (online and printed, images, graphics, films, etc.), in the required quotation and citation style for this work.

---

---

**Standard of the Evaluated Competency**

---

**Statement of the evaluated competency – Code**

Develop native applications [with a database] – **00SS**

**Evaluated elements of the competency**

1. Analyze the application development project
2. Prepare the computer development environment.
3. Prepare the data [for the database(s)].
4. Generate or program the graphical user interface.
5. Program the application logic.

---

## Instructions

---

- Plagiarism, attempts at plagiarism or complicity in plagiarism during a summative evaluation results in a mark of zero (0). In the case of recidivism, in the same course or in another course, the student will be given a grade of '0' for the course in question. (IPEL – Article 5.16).
-

## For this project, use the following working methods:

- Write about one `//comment` per line of code.
- Write the revision history at the beginning of every C# file (enter the date, your name, your student number and the change(s) made in the file). You can also include few milestones when completed.

Revision history:		
Linus Smith	2021-10-01	Created the monster object
Linus Smith	2021-10-02	Completed the move() method

- Always use indentation in your code.
- Cover all your lines of code with a try/catch.

The exception to this are the one-line getters/setters without validation. For example:

```
private hunter goHunter = null; //the hunter object in the map  
  
public Hunter { get => goHunter ; set => goHunter = value; } //public getter/setter
```

- When naming variables, use camelCase notation or underscores, but use only one of these two methods. Also specify in the variable name when a variable was passed as a parameter or exists globally (for example psName means a **p**arameter **s**tring variable for the **n**ame). Always initialize variables.
- Use constants instead of hard-coding.
- Use functions to avoid copy/paste of your code.
- Debug and test your code before submitting it (F5/F9/F10/F11).
- Use Git to save your project. Commit at the end of each day of work with a commit message.
- All the setters which are validating the data should return an appropriate validation message without throwing an exception. These validation messages must always be used correctly by the calling code.
- Do not plagiarize. This is not a group project. Never copy/paste source code from/to anyone.

## Project description

You have to create a Git repository containing a solution and 3 project which implement the MVC (Model-View-Controller) design. The first project is a DLL (Dynamic Link Library) to code the logic for a video game. This DLL will be used to play in Console mode (text-based, second project) and in Windows Forms mode (Graphical User Interface (GUI), third project). There will be differences between the 2 games but they will share a common logic. Code all the objects and common code in the DLL, and code all the view/controller specific code in the Console and the Windows Forms applications.

The video game will be called **Monster Hunter**. In this game, a hunter will have to escape a labyrinth without being killed by monsters.

### DLL

The DLL should contain the following objects.

**MAP** Create an object named **map**. It will be used to detect what happens when the objects (the monsters, the hunter, etc.) move in the map. The map class should contain the following requirements :

- Two variables to hold the width and the height of the map. Write the public getters and the private setters for these properties. The height can never be more than 25 and the width 80. They can also never contain a negative value.
- A one-dimension array to hold all the name (string) of all the .map files (text files) available in the current folder. The getter should be public but the setter should be private.
- The constructor should not accept parameters but it should check for all .map files in the current folder. When it finds them, it should fill the one-dimension array created earlier to make them available to the user interfaces.
- A two-dimension char array to hold the data contained in the selected map file. Make it public.
- A private method to **load** a .map file (this is a text file). The method should accept the filename (string) for the map, a hunter object (described below) and a monsters plural object (described below) as parameters. The map files can contain the following elements :

Contents of the file:	Elements found in a map:
##### #####G          M  # #####  # #####p      ##  # #         ##### ## # # ###M          ## # # ### ##### # #H###          w  # #####  #####	<ul style="list-style-type: none"><li>• # are the walls which cannot be crossed</li><li>• <b>H</b> is where the <b>H</b>unter starts</li><li>• <b>G</b> is the <b>G</b>oal. If the hunter reaches this point he/she wins</li><li>• <b>M</b> is where there are <b>M</b>onsters</li><li>• <b>w</b> is a <b>s</b>word used to attack monsters</li><li>• <b>h</b> is a <b>s</b>hield used to protect from monster attacks</li><li>• <b>p</b> is a <b>p</b>otion used by the hunter</li><li>• <b>x</b> is a <b>p</b>ickaxe used to break the walls</li></ul>

The method which loads the map should open the file and loop in it. It should then copy each character found into the two-dimension char array created earlier. Note that the map may vary in size in the file (20x30, or 15x50, etc.). At the end of the loops which reads the entire .map file, calculate the height and width of the map and set them in the two variables created earlier for this. If the validation fails, make sure the problem is noticed on the view (user interface). However, beside the map size, there is no need to detect or validate any error or inconsistencies in the map. Just make sure that your project includes 3 (three) valid .map files in the bin folder (with the .exe and .dll) and make sure that all the maps you include contain at least some walls (#), a **H**unter starting point, **M**onster(s) and a **G**oal.

If a **M** (monster) is found on the map, create a monster object (described below) and add it to the `monsters` object passed as a parameter. If a **H**unter starting point is found, set the position (X and Y) of the hunter in the object received as a parameter. Then remove all the **P** and **M** from the two-dimension map because these characters will move regularly.

**RANDOM** For events described later, create a singleton random class to generate random numbers.

**SWORD** Create a sword object with these requirements :

- A property for the strength (or offense) of the sword. It will be used to determine the damage inflicted by the sword during an attack (minimum 5, maximum 10). This value should be set randomly when the object is created.
- Note that after an attack, each sword has 1 chance out of 5 to break after an attack (the damage inflicted will still count). So find a place to include this logic somewhere in this object, and return whether the sword did break or not after an attack.

**SHIELD** Create a shield object with these requirements :

- A property for the armor (or defense) of the shield. It will be used to determine the damage protection granted during an attack (minimum 3, maximum 6). This value should be set randomly when the object is created.
- Note that after an attack, each shield has 1 chance out of 4 to break after an attack (the armor will still count). So find a place to include this logic somewhere in this object, and return whether the shield did break or not after an attack.

**PICKAXE** Create a pickaxe object with these requirements :

- Note that after being used, each pickaxe has 1 chance out of 3 to break after breaking a wall (the wall destruction will still be valid). So find a place to include this logic somewhere in this object, and return whether the pickaxe did break or not after being used.

## POTION

Create a potion object with these requirements :

- A property to hold the type of potion found. Create an enumeration that you will use in the property type. There are 4 types of potions (more details on potions effects below):
  - Strength potion : increases the hunter attack/defense. It also restores the missing HP.
  - Poisoned potion : decreases the hunter attack/defense and speed (freeze time).
  - Invisibility potion : makes the hunter able to walk through monsters without engaging in a fight.
  - Speed potion : makes the hunter moves 2 times faster.
- The constructor should select a potion type automatically with a « roll of dice 6». 1=potion. 2-3=speed potion, 4-5 invisibility potion. 6=strength potion.

## CHARACTER

To manage the hunter and the monsters, create a base object named **character**. It should contain the following properties, methods and requirements:

- This object cannot be instantiated (with « var = new character() »). It can only be inherited.
- 2 properties to hold the X and Y position on the screen. These variables must be validated against the maximum X and Y properties (or map width/height). If these properties are still to zero, it means the map is currently being loaded. In that case allow the new values because they will be validated later.
- 2 properties to hold the width and the height of the map (or, if you prefer, of the max X and Y positions where the characters can go). These values should always be equivalent to the size of the map used. For example if we have a map of 12x8, the characters can travel to the very end of the map (even if there is no wall) but they cannot go farther. When setting these values, check if there are already position X / Y values set in the current object. If yes, perform the validation to detect if the new X and/or Y position are outside of the map.
- 2 properties to hold the **maximum HP** (max 20) and the current **HP** of the character (not more than the maximum HP).
- 2 properties to hold the **strength** (or attack) of a character (maximum 7), and the **armor** (or defense) of a character (maximum 4).
- 1 property to hold the number of milliseconds to wait before the character can move again (freeze time).
- 1 abstract method to move the character. It should have the following parameters(int X, int Y) and return a bool.

- To make sure we always have the position for a character, the constructor should accept 2 mandatory values (X and Y) to initialize the object properties. It should also accept 2 optional values (maximum X and Y, with a default value of zero) in case we already know the size of the map.
- One method to check if the character is dead (HP lower or equal to zero).

**HUNTER** To manage the hunter, create an object named **hunter**. It should have the following requirements:

- It should inherit from the character object.
- It should have a name property which cannot hold more than 20 characters.
- A property to hold the score of the hunter (maximum 100 000, no negative value)
- A property to hold the state of the character. Use an interface and 5 classes to implement it. To the exception of the Normal state, all the other states are the result of drinking a potion. Because every potion lasts for only 10 seconds, change the hunter attributes (strength, armor, etc.) in the setters only. The state can be :
  - **Normal** : This is the default state for the hunter which uses its normal attributes.
  - **Strong** : This occurs when a hunter drinks a strength potion. In that case his/her strikes do twice (2 times) the calculated damage, and his/her defense is 1.5 times stronger. Also, all his/her wounds are healed and the HP is restored to the maximum HP.
  - **Poisoned** : When a hunter drinks a poisoned potion, his/her HP is dropped by 5. Also his/her attack and defense are decreased by 50%. His/her freeze time is also increased by 25%.
  - **Invisible** : In that state, the hunter can walk over the monsters, and the monsters can walk over the hunter without engaging in a fight.
  - **Fast** : In that state, the hunter freeze time is decreased by 50%.
- A method to move the hunter. The hunter can only move where there is no wall and no monster. When a hunter moves, change immediately the position of the hunter (X, Y).
- The constructor should set the hunter freeze time to 1 second. It should also receive a mandatory position (X, Y) and it should pass it to the base object constructor.

**MONSTER** To manage every monster, create a **monster** object. It should have the following requirements:

- It should inherit from the character object.
- A method to move the monster. The monster can only move where there is no wall and no hunter. When a monster moves, change immediately its position (X, Y).
- The constructor should set the monster freeze time to 2 second. It should also receive a position (X, Y) and it should pass it to the base object constructor.

**MONSTERS** To manage all the monsters in a map, create a List<> of monster objects with this element :

- Create a method to find one or more monsters by its position (X, Y). Return all the singular monster objects that are present at this position (you can return a List<>, an array, etc).

## CONSOLE

When starting the console program, it should first validate the name of the player. It should loop until the name is valid.

It should then show the list of all availables maps and let the user choose one by entering a number. For example :

**1 : Castle.map**  
**2 : Marsh.map**  
**3 : Hell.map**

Loop until a valid choice is made. When the map is selected, draw it in the Console (like it appears in the file) and start the game. Note that the hunter (the H letter) and the monsters (the M letters) should always be displayed in color.

Anywhere on the screen, and in any order or position, write the following information board :

<b>Player: Hot Diggity Dogger</b>	<b>Map: Castle.map</b>
<b>HP : 20</b>	<b>Level : 1</b>
<b>Score : 0</b>	<b>Infos :</b>

The infos field should always display at least the 3 last action that took place on the map, like:

- The hunter found a shield.
- The hunter does 7 of damage to the monster (5 HP remaining).
- The monster does 3 of damage to the hunter.

For the next specifications, you can add any property or method to the objects if you think it makes sense to place the requirements in the Controller.



## Monsters movement

When the game begins, start a thread to move all the monsters accordingly to their freeze time (2 seconds by default). So at every 2 seconds, move every monster alive on the map by one square. Choose randomly where the monsters (the M letters) go (up/down/left/right). If there is a blank space anywhere beside a monster it must move at every 2 seconds, it cannot stay on its square. It is OK if 2 monsters are on the same spot. It is also OK if the monsters walk over the potions, just make sure we can always see the monsters on the screen. When the monsters then move, we should be able to see the potion again.

When the monsters check for a move, if the hunter is on a square right beside a monster, they must not move but attack the hunter. When attacking, take the strength/attack of the monster (for example 8) and subtract the armor (for example 5). The result is the amount of damage inflicted (for example  $8 - 5 =$  damage of 3). If the hunter has a shield, add the armor/defense value to the defense of the hunter. After the shield was used, it has 1 chance out of 4 to break. If it breaks, the hunter loses it (write it in the infos). The strength and armor of the hunter can also be affected by the potions (handled automatically by the setters which should always check the state of the hunter). In the infos field, write the number of damage inflicted to the hunter (or if the hunter died). Also update the HP on the information board. If the HP is 5 or below, write the number in red. If the hunter died during this attack, write it in the Infos field and ask the user to play another game. If the answer is (y)es, ask again for the name and map and start a brand new game (Level 1, Score : 0, etc.). If the answer is (n)o, leave the program.

If the hunter did not died during the attack, he/she attacks back with the same formula (strength of hunter – armor of monster). The sword, shield and potions affects the fights as explained earlier. In the infos field, write the damage inflicted and the remaining HP of this monster (or if the monster died). If the hunter attacks with a sword, it has 1 chance out of 5 to break after the hit. If it breaks, the hunter loses it (write it in the infos).

## Hunter movement

Independently of the 2 seconds thread which handles the monsters movements, the user can hit keys to move up/down/left/right by one square. Use the keys you want (WASD, or arrows (↑→↓←), or any other keys) but tell them to the user before the game is started. If the hunter tries to move toward a wall or outside of the map limits, nothing should happen (no movement, no freeze time). However if the hunter moves on an empty space or on a space where there is an item, it should move immediately its position, but it should then wait for the freeze time (for example 1 second) before being able to move again. If the hunter walks on a sword/shield/pickaxe it will replace any existing sword/shield/pickaxe the hunter already has. The player will also get 50 points for getting such an item. If the hunter walks on a potion, its random effects should replace the effects of any potion previously found. For every potion drank, change the color of the hunter for a specific color unique to each potion (so the hunter can have 5 different colors, one for each state). Every potion drank by the hunter should add 25 points to the score.

If the hunter tries to move on a square where there is a monster, he/she will not move but will rather attack all the monsters present on the square (use the method created earlier to find them). The fighting rules are exactly the same than described earlier except that this time the hunter attacks first. The sword/shield/potions should also have the same behavior than described earlier. The hunter gets 100 points to defeat a monster, so update the score accordingly.

If the hunter is still alive after the fight (which can involve many monsters), wait until the freeze time is completed before the hunter can move or attack again. If the hunter dies during a fight, ask the player to play again like described earlier.

If the hunter goes toward a wall and has a pickaxe, he/she will break it and it will become a free square on which the hunter will move (freeze time will then apply). However, after the pickaxe is used, it has 1 chance of out 3 to break. So if it breaks, the hunter loses it (as usual, write all the events in the Infos field).

If the hunter reaches the Goal, add 500 points to the score and wait at least 3 seconds to let the user read the new score. After that delay :

- Increase the Level by 1 (ex : Level 2)
- The monsters freeze time should be decreased by 10% compared to the previous level.
- The hunter freeze time should be decreased by 5% compared to the previous level.
- Increase the HP of the hunter by half the missing points required to reach the maximum HP (for example if the hunter has a maximum HP of 20 and a current HP of 10, restore the HP to 15. If the HP is below the maximum, always increase the HP by at least 1 point, so a hunter with 19 of HP would have an increase of 0.5 but it would not make sense so increase it to 20.
- Reload the map from the file with all its objects, walls and characters (only the hunter keeps its properties, HP, weapons, etc.). Do not reset the information board (HP, score, etc.).

Then continue the game which should be faster and harder as the player moves to the next levels.

## Game over

When the hunter dies the game is over. Save the **name** and the **score** of the player in a file to hold all the top scores. The file should contain a maximum of 10 lines, and should be ordered from the highest score to the lowest one, like :

Samantha	14000
Linus	8900
Julie	7000
Steve	5775
Miranda	650
Bill	25


So it is possible that a score is not saved to the file if the high scores are all higher than the latest score. So make sure you always show the latest score to the player even if it is not saved in the file.

Then ask the player to start a new game.

## WINDOWS FORMS

The Windows Forms game uses exactly the same logic described in the Console application. It also uses all of the requirements. Here are the key differences between the 2 versions :

- When asking a player name and map for a new game, use a **Modal** form. This form should use a textbox for the player name and a combobox to select a map among all the .map files found. It should also have a OK/Save button and a Cancel button. When clicking OK/Save, the validation messages should be displayed in red beside each field (for example : the name cannot contain more than ? characters, etc.). If all the information is valid, the form should close and the entered infos should be passed to the main form to start the game. If the user clicks Cancel, the modal form should close and then the Main form should close too.



The screenshot shows a modal dialog box with a light gray background. It contains two input fields: 'Player name:' with a text box containing 'Hot Diggity Dogger!' and 'Map:' with a combobox showing '.\Castle.map'. Below these fields are two buttons: 'Start Game' and 'Quit program'.

- You must use the same two-dimension array to manage the positions of all items and characters. However you must use bigger squares to draw them. In other words, each character of the two-dimension array should be represented by a square of many pixels in the Windows Forms program. All the walls, items and monsters must be represented by a picture. For example the walls represented by a single character (#) in the console game may be represented by a bigger picture here :



So the chars **##p##** can be represented with many pictures with Windows Forms :

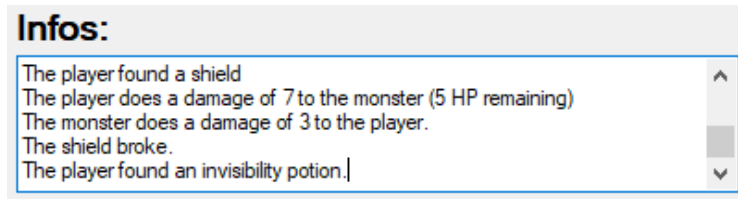


You can also use the Drawing library to draw them manually instead of using pictures.

Note that, compared to the Console game, all the coordinates (X, Y) will be exactly the same in memory. It is only the display which is different.

- Because the hunter must change its color with the potions/states, you must use the Drawing library to draw the hunter. The picture does not need to be elaborate. It can be only a face, a stickman, etc. As long as it is clear it is a hunter, it is OK.
- Use a **timer** control to trigger the movements of the monsters.

- When the monsters and hunter move, update their position instantly in memory. However, use a **thread** and a **delegate** to move them for the duration of the freeze time. For example it could take one second for the hunter to go from square 1,3 to the square 2,3 (update the animation at least 10 times per second). After this delay the hunter reaches its destination and it can move again.
- For all other requirements related to time, you may use a timer, a thread, an event, or any regular code that you think may fit your needs.
- In the information board, the Infos section should never delete/overwrite any « Infos ». Use a multi-line textbox to include all the events of the level, like :



- In the information board, use a progress bar in addition of the actual HP value, like :



- At the end of the game, show the **High Score** list in a Modeless form. The user may leave this form always open, so don't forget to update it at the end of each game. If a player just made it to the high score list, write his/her name in bold and/or in color to show it is a new entry. There is no need to save this bold/color to the file. This is just a temporary display right after a player did a high score.

## MODEL / VIEW / CONTROLLER

**Try/catch** All the code of the 3 projects (Model, Views/Controllers) should be covered with a try/catch. All the catch blocks should save the exceptions/error message in a .log file (text) by including the **date and time** when the exception occurred, the name of the **class** and **method** where it occurred, and the exception **message** from C#, like:

**2021-10-01 19:46:22** An error occurred in the **move** function of the **hunter** object : **Division by zero**

In the Console and GUI projects, the users should be able to see all the exception message so they will know when something unexpected happened, like : **An error occurred : division by zero**

Points will be awarded to fulfill the asked **requirements**. You may add anything you think is relevant to fulfill the requirements (methods, properties, etc). However, if you only want to create a more beautiful game (sounds, animations, etc.), no points could be awarded for this. So keep in mind that your priority should be the functionalities described in this project. Finally, the sooner you will start this project, the easiest it will be to be ready on time.