INFO 0402 - Projet Bomberman

Table des matières

1 Introduction

Pour le projet d'INFO 0402, il vous est proposé de développer une version simplifié du célèbre jeu des années 1980 Bomberman. La base de ce projet est énoncé dans la première section du TP 1. Cependant, certaines précisions seront apportées dans ce document. Vous pouvez donc ajouter ou supprimer autant de classes que nécessaires par rapport au TP 1. Seul les règles de base sont conservées. À savoir :

— le but du jeu est de détruire tous les ennemis, puis d'atteindre l'objectif,
— le joueur ne peut, de base, déposer qu'une seule bombe à la fois. Il doit attendre que la bombe ait explosé avant de poser une nouvelle bombe,
— il y a trois types d'ennemis disponibles
  1. `Monster` qui sont des ennemis de base.
  2. `Ghost` qui sont des ennemis qui peuvent traverser les murs.
  3. `Bowman` qui sont des ennemis pouvant attaquer à distance.
— différents `Item` peuvent se trouver sur la carte afin d'améliorer les capacités du joueur. Seul le joueur récupère automatiquement un `Item` en allant sur sa case. Les différents `Item` sont
  1. `MoreLife` augmente le nombre de points de vie de notre Bomberman,
  2. `PowerUp` augmente la puissance des bombes,
  3. `MoreBomb` augmente le nombre de bombes,
  4. `SpeedUp` augmente la vitesse du Bomberman,
  5. `ScaleUp` augmente la portée des bombes,
— les `Item` sont invincibles et ne disparaissent que lorsque le joueur le récupère,
— lorsqu'une bombe explose, elle touche les cases voisines horizontalement et verticalement sur une certaine portée. La portée n'est bloquée que par les murs,
— les bombes infligent des dégâts aussi bien aux ennemis, qu'au joueur,
— si une bombe touche une autre bombe, cette dernière explose aussi,
— enfin, certains murs peuvent être invincibles.

Le projet se programmera en C++ avec comme version minimale le standard de 2014. Il sera donc interdit d'utiliser des méthodes de la bibliothèque standard devenues dépréciés ou obsolètes à partir du C++14. Sauf mention contraire, toute autre bibliothèque que la STL sera à proscrire.

Toute l'interface homme-machine se fera avec l'entrée au clavier (`std::cin`) et la sortie sur la sortie standard (`std::cout`).

Le projet pourra se faire à deux. Cependant, il faudra dans ce cas rajouter pour chaque méthode, indiquer sous la forme d'un commentaire (cf. Section 9.6) le ou les auteurs. Ici le but et de pouvoir vous évaluer au plus juste. Par exemple, quelqu'un pourrait faire moins de méthodes, mais des méthodes plus compliquées.

2 Déroulement du jeu

Au démarrage du jeu, un message sera affiché à l'utilisateur, ainsi qu'un menu lui proposant de jouer, d'avoir accès aux règles du jeu et à la légende de la carte ou de quitter.

Afin de gérer le jeu, une classe `System` se chargera d'afficher la carte, de proposer au joueur de jouer, de faire bouger tous les ennemis et de gérer l'explosion des bombes. Entre chaque action, la carte sera actualisée.

Une façon d'afficher la carte serait

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   | w |   | W |   | W |   | W |   | W |   | w |   | W |   |   |
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   | | | P | O |   |   |   |<--| B |   |   | SC| X |   |   |   |
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|-@-|---|-- |   | M | PU| w | I | W |   |   |   | w |   |   |
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   | | | W |   |   | G |   | ML|   | I |   |   | MB|   |   |   |
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   | | |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   | | |   | I |   | I |   | I |   |   |   | SP|   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

où	`P`	Le joueur.
	`O`	Une bombe active.
	`@`	Un bombe qui explose avec les différentes cases touchées. Notez que la dernière case touchée n'est pas entièrement traversée.
	`W`	Un mur intact.
	`w`	Un mur fissuré.
	`I`	Un mur invincible.
	`M`	Un monstre de base.
	`G`	Un monstre de type `Ghost`.
	`B`	Un monstre de type `Bowman`.
	`<--`	Une flèche du `Bowman` allant vers la gauche.
	`ML`	Un `Item MoreLife`.
	`PU`	Un `Item PowerUp`.
	`MB`	Un `Item MoreBomb`.
	`SP`	Un `Item SpeedUp`.
	`SC`	Un `Item ScaleUp`.
	`X`	L'objectif.

Pour visualiser les sens des flèches, nous proposons :

```
   A
   |
<--B-->
   |
   V
```

Libre à vous de proposer tout autre affichage.

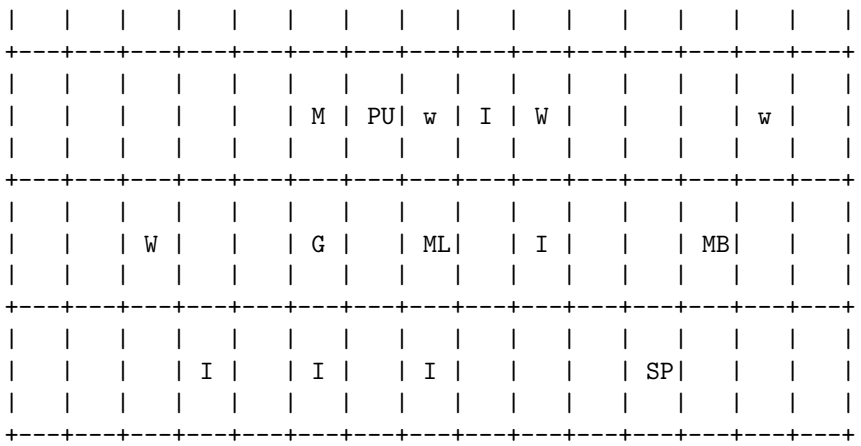
2.1 Chargement d'un niveau

La carte sera préalablement stockée dans un fichier sous la forme de texte. Une façon de faire pourrait être pour chaque case, d'indiquer le type de tuile, et de les séparer (par une virgule par exemple). Par exemple,

```
,w,,W,,W,,W,,w,,W,
,,P,,,,,B,,SC,X,,,,
,,,,,M,PU,w,I,W,,,,w,
,,W,,,G,,ML,,I,,,MB,,
,,,I,,I,,I,,,,SP,,,
```

donnerait le niveau suivant

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   | w |   | W |   | W |   | W |   | W |   | w |   | W |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   | P |   |   |   |   |   | B |   |   | SC| X |   |   |   |
```



3 Action du joueur

Afin de pouvoir jouer, un choix sera proposé à l'utilisateur entre se déplacer ou poser une bombe. Par exemple, l'utilisateur pourrait taper

- `8` pour aller vers le haut,
- `4` pour aller vers la gauche,
- `6` pour aller vers la droite,
- `2` pour aller vers le bas,
- `5` pour poser une bombe.

Si le déplacement est impossible, il sera à nouveau proposé au joueur l'action qu'il veut faire. Si l'utilisateur pose une bombe, il pourra se déplacer directement.

4 Comportement des bombes

Une bombe sera disposée à l'emplacement du joueur. Elle attendra trois tours avant d'exploser. Enfin, seul les `Ghost` auront la capacité de traverser une bombe.

5 Déplacement des `Ghost`

Les `Ghost` pouvant se déplacer à travers les bombes et les murs, il sera proposé de les afficher sur la même case que ceux-ci lorsqu'ils les traversent. Si une bombe touche un mur ou une autre bombe qui contient un `Ghost`, alors ce `Ghost` est aussi touché (deux fois dans le cas de la bombe : l'originale puis celle qui est touchée).

6 Déplacement des ennemis

Un algorithme simple sera utilisé pour déplacer les ennemis : ils essaieront d'abord de se mettre sur la même ligne, puis sur la même colonne. Si un mur est sur le chemin de l'ennemi, il passera son tour. Enfin, si un ennemi touche le joueur, cet ennemi passera 3 tours pour laisser le temps au joueur de partir.

7 Attaque des `Bowman`

Lorsqu'un `Bowman` voit le joueur (il n'y a ni mur, ni ennemi entre lui et le joueur), il tire une flèche vers le joueur. Cette flèche se déplacera d'une case de façon linéaire par tour.

8 Fin du jeu

Lorsque le joueur a gagné (tous les ennemis sont détruits et il a atteint l'objectif) ou perdu (il n'a plus de vie), le jeu affiche un message correspondant au type de fin, puis le menu de démarrage.

9 Améliorations bonus

Les améliorations proposées ici ont pour but d'ajouter des points bonus à la note finale, afin de rattraper des points qui ne seraient pas complets sur toute l'implémentation. Cependant, la plupart des améliorations nécessite une base bien consolidée du projet.

9.1 Amélioration de l'affichage

Affichez le nombre de points de vie sous forme de cœurs au-dessus de la carte.

9.2 Amélioration des `Item`

— Rajoutez un `Item ControlBomb` permettant de rajouter une option au joueur : choisir quand toutes les bombes explosent en même temps.
— Rajoutez un `Item GhostMode` permettant au joueur d'avoir le même comportement que les `Ghost`.

9.3 Cacher l'objectif

Rajoutez le fait que l'objectif ne soit pas visible de base. Il prendra l'apparence d'un mur destructible, et l'apparence de l'objectif, uniquement lorsque le mur sera détruit.

9.4 Plusieurs niveaux

Rajoutez plusieurs niveaux : lorsque le joueur termine un niveau, un niveau différent est chargé.

9.5 Mode deux joueurs

— Rajoutez la possibilité de jouer à plusieurs. Les deux joueurs joueront à tour de rôle, puis ce seront les ennemis et les bombes qui s'activeront.
— Rajoutez un mode de jeu où les joueurs jouent l'un contre l'autre (avec ou sans ennemi).

9.6 Documentation Doxygen

Rajoutez une documentation de type Doxygen à votre code. Doxygen est un logiciel qui analyse le code source à la recherche de commentaires formatés. Ces commentaires sont ensuite transformés sous différents formats dont le site Web (le plus utilisé) ou encore sous forme de code LaTeXpouvant être compilé en PDF (cf. Annexe A). Plusieurs écritures de documentation Doxygen sont possibles, mais une seule sera présentée ici. Les documentations se font au-dessus des prototypes de fonctions et en début d'en-tête. Conventionnellement, on ne documente pas les fichiers sources en mode Doxygen.

Tout commentaire à prendre en compte par Doxygen s'écrit sous la forme

```
/**
 * ...text...
 */
```

avec autant de ligne que nécessaire. Les `*` intermédiaires en début de ligne sont optionnels.

Pour inclure un fichier dans la documentation, celui-ci devra commencer par

```
/**
 * \file nom_du_fichier
 */
```

Une description brève du fichier ou d'une méthode se fait avec `\brief`, tandis qu'une description longue se fait en revenant à la ligne.

```
/**
 * \file nom_du_fichier
 * \brief Description breve du fichier
 *
 * La description longue est ici.
 */
 class Foo
 {
 public:
    /**
```

```
     * \brief Une methode
     *
     * Une methode qui ne fait pas grand chose.
     */
    void uneMethode();
};
```

Pour ajouter des auteurs, nous utilisons la balise `\author`. La liste des paramètres d'une fonction se fait avec des balises `\param`, et le retour (si ce n'est pas `void`) avec `return`. Il ne faut pas indiquer les types des paramètres et du retour.

```
/**
 * \brief Une petite methode
 * \author John Doe
 * \param a le premier parametre
 * \param b le deuxieme
 * \return la valeur calculee
 *
 * Une petite methode qui fait un calcul bien savant.
 */
int calcul(int a, char b);
```

Il est possible de commenter des structures en dehors de celles-ci.

```
/**
 * \class Foo
 * \brief Une classe d'exemple
 * \author Quelqu'un
 *
 * Une classe qui ne sert vraiment que d'exemple
 */

 /**
  * \enum e_type
  * \brief Un exemple d'enumeration
  * \author Quelqu'un d'autre
  *
  * Un exemple d'utilisation pour l'enumeration
  */


class Foo
{
public:
    enum e_type
    {
        RIEN
```

```
    };
};
```

Les différentes balises de structures sont
- `\struct` pour les structures C,
- `\union` pour les unions,
- `\enum` pour les énumérations,
- `\fn` pour les fonctions,
- `\var` pour les valeurs de variables, de `typedef` ou d'énumérations,
- `\def` pour les `#define`,
- `\typedef` pour les `typedef`,
- `\file` pour les fichiers,
- `\namespace` pour les espaces de nom.

Ces différentes balises ne sont à utiliser que si le commentaire n'est pas juste au-dessus de la structure commentée.

Pour documenter des attributs de classe, il faut utiliser `/**< ... */`.

```
class Foo
{
private:
    int m_att; /**< un entier */
};
```

Afin d'écrire des listes dans les descriptions longues, il est possible d'utiliser les balises de listes HTML.

Enfin, pour tout autre formatage (titres, tableaux etc.) vous pouvez vous reporter à la documentation du Doxygen.

10 Rendu et rapport

10.1 Rendu du projet

La date de rendu du projet sera déterminée plus tard, mais sera indiquée sur la section projet du Moodle.

Le rendu du projet se fera sur le Moodle, dans la catégorie projet, sous la forme d'une archive. Aucun document envoyé par mél ne sera accepté. Vérifiez donc bien que vous avez accès au rendu et que celui-ci fonctionne en amont de la date de rendu afin de pouvoir signaler tout problème.

Le projet en lui-même ne devra contenir que le code source (en-têtes et sources), ainsi qu'un ou plusieurs Makefile permettant de le compiler, et éventuellement le Doxyfile (et non la documentation générée) si vous avez fait une documentation Doxygen (Section 9.6). Il ne devra en aucun cas y avoir de fichiers propre à un IDE (`.pro`, `.vproj` etc.).

10.2 Rapport du projet

Le projet sera accompagné d'un rapport au format PDF. Ce rapport devra contenir au minimum :

1. une première partie concernant la conception
   (a) une présentation du projet,
   (b) votre description formelle du projet,
   (c) les différents diagrammes que vous aurez produit,
2. une seconde partie concernant la partie développement
   (a) pour chaque concepts de la POO et du C++ non triviaux, expliquer
      — ce que sont ces concepts,
      — comment ils fonctionnent,
      — pourquoi vous avez eu besoin de les utiliser.
3. si vous avez fait la partie sur la documentation Doxygen (Section 9.6), celle-ci pourra être mise à côté en version PDF.

A Utilisation de Doxygen

Afin d'utiliser Doxygen pour générer la documentation du code, il faut l'utiliser sur un fichier appelé (par convention) Doxyfile. Ce fichier contient toutes les instructions de formatage des documentations. Afin de ne pas avoir à gérer ce fichier manuellement, vous pouvez utiliser l'outil Doxywizard qui est un outil graphique de gestion de Doxyfile. Il faudra donc installer Doxygen et Doxywizard.

Comme nous pouvons le voir sur la Figure 1, il faut indiquer le répertoire de travail.

Doxywizard est découpé en trois onglets : `Wizard` qui contient les informations les plus courantes, `Expert` pour spécifier un peu plus le format de la documentation et `Run` pour lancer la génération de la documentation.

A.1 Onglet `Wizard`

Dans l'onglet `Wizard`, *topic* `Project` (Figure 1), nous pouvons spécifier le nom du projet, qui déterminera les noms du site Web ou du PDF. Il faudra aussi sélectionner le répertoire du projet, et cocher l'option pour analyser récursivement au besoin. Enfin, il faut préciser le répertoire où sera générée la documentation.

Dans le *topic* `Mode` (Figure 2), nous pouvons indiquer si l'on souhaite intégrer ce qui n'est pas commenté et le langage du projet.

Dans le *topic* `Output` (Figure 3), nous pouvons choisir le format généré. Lorsque l'on choisit de faire une version LaTeX, un Makefile est généré, et le PDF se fait avec `make pdf`. Attention, il est nécessaire d'avoir un environnement de compilation LaTeX(`pdflatex` par exemple).

Dans le *topic* `Diagrams` (Figure 4), nous pouvons choisir le format des diagrammes d'inclusions. Ces diagrammes permettent de voir pour chaque classe ce qu'elle inclut, et ce de façon récursive. Ces diagrammes permettent aussi de naviguer entre les différentes classes.

A.2 Onglet `Expert`

L'onglet `Expert` ne sera pas détaillé ici. Vous pouvez vous reporter à la documentation.

A.3 Onglet `Run`

L'onglet `Run` permet de lancer Doxygen avec toutes les informations contenues dans le Doxyfile généré par les onglets `Wizard` et `Expert`.

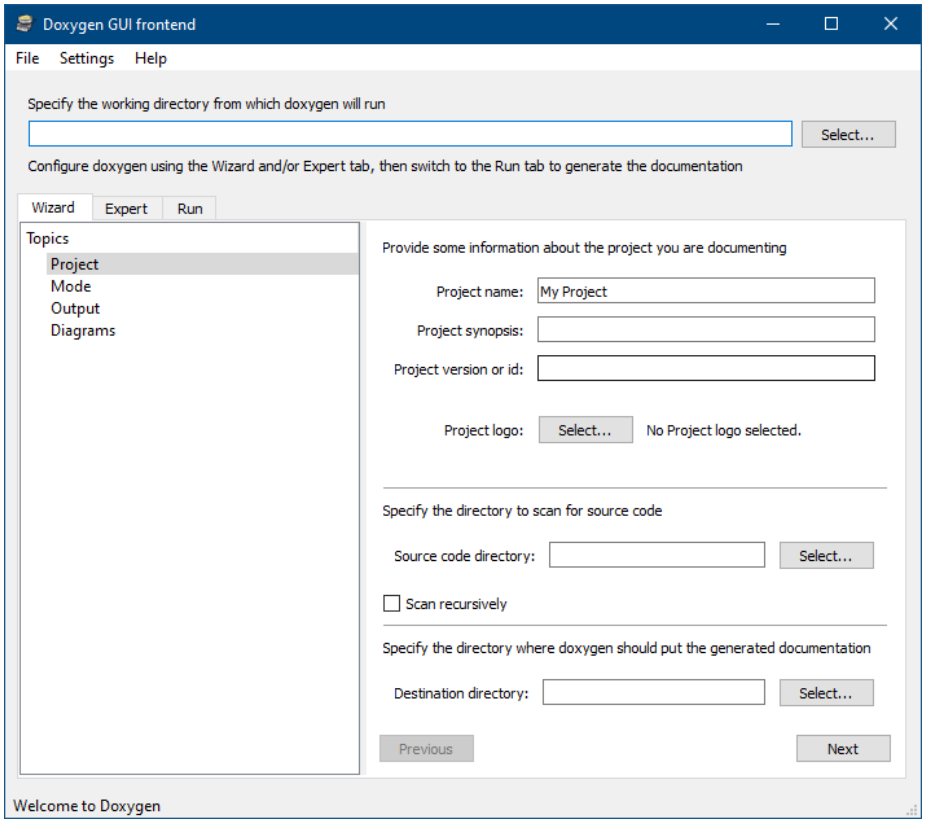


FIGURE 1 – Interface de Doxywizard

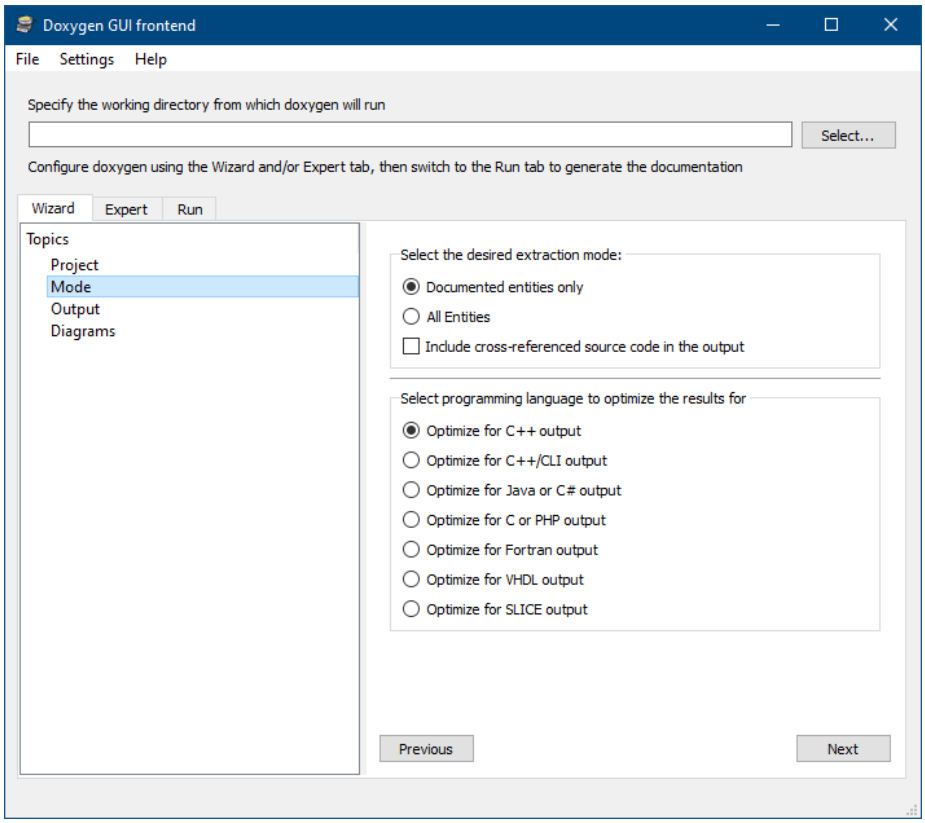


FIGURE 2 – *Topic* `Mode`

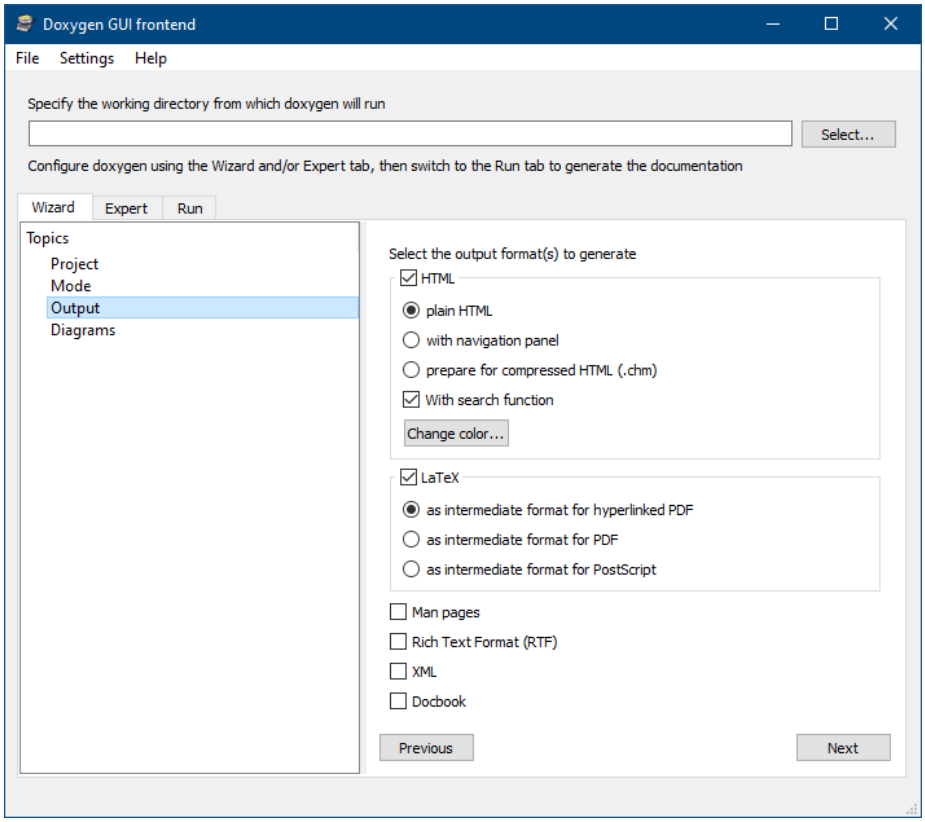


FIGURE 3 – *Topic* `Output`

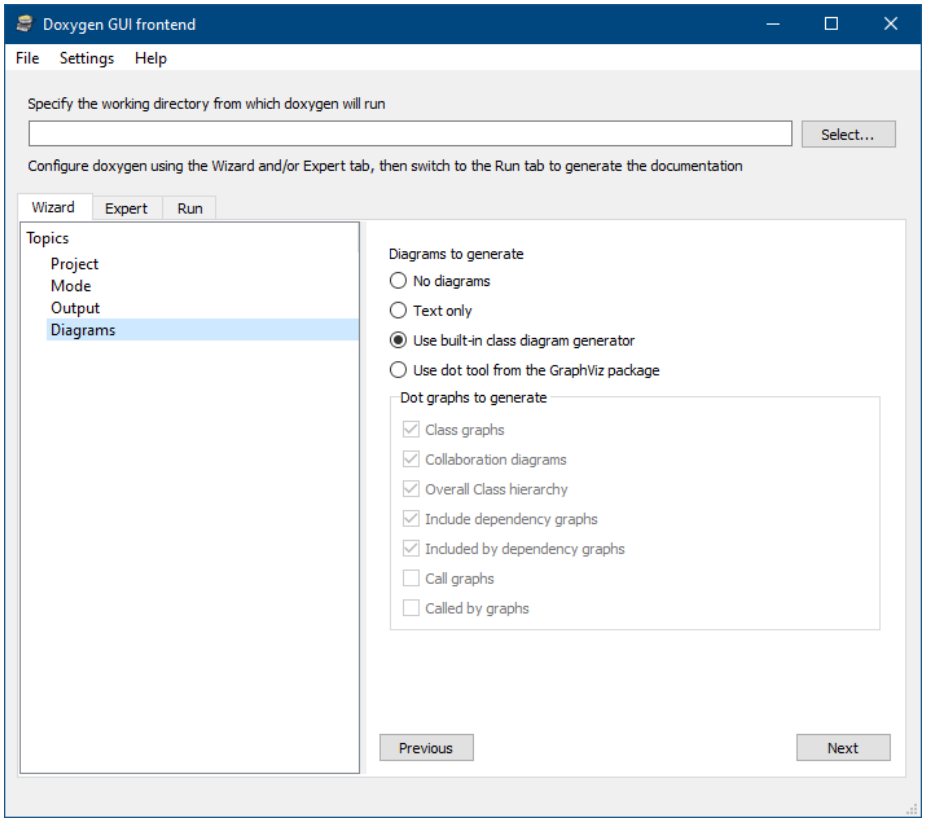


FIGURE 4 – *Topic* `Diagrams`