Interpreter opisu działań obiektów – Etap 1.

1 Wprowadzenie

Niniejszy dokument opisuje zakres prac, które należy wykonać w ramach 1. etapu. Etap ten nominalnie trwa od najbliższych zajęć do zajęć następnych. Jednak do najbliższych zajęć należy wykonać wstępne prace, które wymienione są w dalszej części niniejszego opisu.

2 Zakres prac

W ramach pracy nad programem w całym 1. etapie należy zrealizować następujące podzadania:

- Stworzenie czterech wtyczek, które będą ładowane przez program w momencie startu.
- Zaprojektowanie i zdefiniowanie struktur danych, które będą pozwalały łączyć nazwę polecenia, z funkcjami interfejsu wtyczki obsługującej dane polecenie.
- Rozwijanie makr preprocesora w czytanym pliku *programu działań*.
- Wywołanie funkcji właściwej wtyczki dla wczytanego polecenia z pliku działań i wczytanie parametrów działania.
- Wyświetlanie parametrów działania.

Program powinien poprawnie wczytywać uproszczoną sekwencję poleceń bez słów `Begin_Parallel_Actions` i `End_Parallel_Actions`.

3 Zakres prac wstępnych

Wymienione poniżej podzadania należy zrealizować do najbliższych zajęć. Stanowią one część zadań wymienionych we wcześniejszym rozdziale. Wspomniane podzadania to:

- Bazując na dostarczonym zalążku należy stworzyć drugą *wtyczkę* i ją poprawnie skompilować oraz skonsolidować. Podobnie jak w przypadku pierwszej wtyczki, należy ją załadować w funkcji `main` oraz uruchomić odpowiednie elementy jej interfejsu, tak jak to ma miejsce w przypadku wspomnianej pierwszej *wtyczki*.
- Rozwijanie makr preprocesora w czytanym pliku *programu działań* i wyświetlenie zawartości pliku po przetworzeniu go przez preprocesor języka C.

4 Struktury danych

W programie powinny być zdefiniowane klasy dziedziczące następujące klasy abstrakcyjne:

- klas abstrakcyjna `AbstractInterp4Command`, która definiuje interfejs wtyczki i jest klasą bazową dla klas interpreterów poleceń definiowanej dla każdej z wtyczki,

- klasa abstrakcyjna `AbstractMobileObj`, która jest klasą bazową dla typów obiektów mogących pojawić się na scenie.
- klasa abstrakcyjna `AbstractScene`, która zawiera kolekcję obiektów sceny.
- klasa abstrakcyjna `AbstractComChannel`, która jest klasą bazową dla klasy zapewniającej bezpieczną komunikację z serwerem poprzez gniazdo sieciowe.

Wyżej przedstawione klasy abstrakcyjne definiują niezbędne interfejsy.

4.1 Klasa abstrakcyjna `AbstractInterp4Command`

Klasa abstrakcyjna definiująca interfejs wtyczek. Jest ona klasą bazową dla klas implementujących interpretery poleceń w poszczególnych wtyczkach.

```
class AbstractInterp4Command {
 public:
  /*!
   * \brief Wyświetla postać bieżącego polecenia (nazwę oraz wartości parametrów)
   */
   virtual void PrintCmd() const = 0;
  /*!
   * \brief Wyświetla składnię polecenia
   */
   virtual void PrintSyntax() const = 0;
   /*!
    * \brief Wyświetla wartości wczytanych parametrów
    */
   virtual void PrintParams() const = 0;

   /*!
    * \brief Udostępnia nazwę polecenia
    */
   virtual const char* GetCmdName() const = 0;
   /*!
    * \brief Wykonuje polecenie oraz wizualizuje jego realizację
    *
    * Wykonuje polecenie oraz wizualizuje jego realizację.
    * \param[in,out]  rScn - scena zawierającą obiekty mobilne,
    * \param[in]      sMobObjName - wskaźnik na nazwę lokalizującą i identyfikującą obiekt,
    * \param[in,out]  rComChann - kanał komunikacyjny z serwerem graficznym.
    * \retval true - operacja powiodła się,
    * \retval false - w przypadku przeciwnym.
    */
   virtual bool ExecCmd(AbstractScene      &rScn,
                        const char         *sMobObjName,
                        AbstractComChannel &rComChann) = 0;
   /*!
    * \brief Czyta wartości parametrów danego polecenia.
    *
    * Czyta ze strumienia wartości parametrów polecenia.
    * \param[in] rStrm_CmdsList - strumień, z którego są czytane parametry polecenia.
    * \retval true - operacja powiodła się,
    * \retval false - w przypadku przeciwnym.
    */
   virtual bool ReadParams(std::istream &rStrm_CmdsList) = 0;
};
```

4.2 Klasa abstrakcyjna `AbstractMobileObj`

Klasa stanowiąca składnik interfejsu wtyczek. Jest ona klasą abstrakcyjną definiującą interfejs do typów obiektów mobilnych, które mogą pojawić się na scenie. Moduł klasy `Vector3D` jest dostarczony w nowej wersji zalążka do zadania.

```
/*!
 * Zakładamy, że przód obiektu jest wskazywany przez strzałkę osi OX.
 * Nazwy metod są obowiązujące.
 */
 class AbstractMobileObj {
  public:
    /*!
     * \brief Udostępnia wartość kąta \e roll.
     *
     *  Udostępnia wartość kąta \e pitch reprezentuje rotację
     *  zgodnie z ruchem wskazówek zegara wokół osi \e OX.
     *  \return Wartość kąta \e roll wyrażona w stopniach.
     */
    virtual double GetAng_Roll_deg() const = 0;
    /*!
     * \brief Udostępnia wartość kąta \e yaw.
     *
     *  Udostępnia wartość kąta \e pitch reprezentuje rotację
     *  zgodnie z ruchem wskazówek zegara wokół osi \e OY.
     *  \return Wartość kąta \e pitch wyrażona w stopniach.
     */
    virtual double GetAng_Pitch_deg() const = 0;
    /*!
     * \brief Udostępnia wartość kąta \e yaw.
     *
     *  Udostępnia wartość kąta \e yaw reprezentuje rotację
     *  zgodnie z ruchem wskazówek zegara wokół osi \e OZ.
     *  \return Wartość kąta \e yaw wyrażona w stopniach.
     */
    virtual double GetAng_Yaw_deg() const = 0;

    /*!
     * \brief Zmienia wartość kąta \e roll.
     *
     *  Zmienia wartość kąta \e roll.
     *  \param[in] Ang_Roll_deg - nowa wartość kąta \e roll wyrażona w stopniach.
     */
    virtual void SetAng_Roll_deg(double Ang_Roll_deg) = 0;
    /*!
     * \brief Zmienia wartość kąta \e pitch.
     *
     *  Zmienia wartość kąta \e pitch.
     *  \param[in] Ang_Pitch_deg - nowa wartość kąta \e pitch wyrażona w stopniach.
     */
    virtual void SetAng_Pitch_deg(double Ang_Pitch_deg) = 0;
    /*!
     * \brief Zmienia wartość kąta \e yaw.
     *
     *  Zmienia wartość kąta \e yaw.
     *  \param[in] Ang_Yaw_deg - nowa wartość kąta \e yaw wyrażona w stopniach.
     */
```

```
   virtual void SetAng_Yaw_deg(double Ang_Yaw_deg) = 0;

   /*!
    * \brief Udostępnia współrzędne aktualnej pozycji obiektu.
    *
    * Udostępnia współrzędne aktualnej pozycji obiektu
    * \return Współrzędne aktualnej pozycji obiektu. Przyjmuje się,
    *         że współrzędne wyrażone są w metrach.
    */
   virtual const Vector3D & GetPositoin_m() const = 0;
   /*!
    * \brief Udostępnia współrzędne aktualnej pozycji obiektu.
    *
    * Udostępnia współrzędne aktualnej pozycji obiektu
    * \return Współrzędne aktualnej pozycji obiektu. Przyjmuje się,
    *         że współrzędne wyrażone są w metrach.
    */
   virtual void SetPosition_m(const Vector3D &rPos) = 0;

   /*!
    * \brief Zmienia nazwę obiektu.
    *
    *  Zmienia nazwę obiektu, która go identyfikuje.
    *  \param[in]  sName - nowa nazwa obiektu.
    */
    virtual void SetName(const char* sName) = 0;
   /*!
    * \brief Udostępnia nazwę obiektu.
    *  Udostępnia nazwę identyfikującą obiekt.
    *  \return Nazwa obiektu.
    */
    virtual const std::string & GetName() const = 0;
};
```

4.3 Klasa abstrakcyjna `AbstractScene`

Klasa zawierająca kolekcję poszczególnych obiektów mobilnych.

```
class AbstractScene {
   public:
    virtual AbstractMobileObj*  FindMobileObj(const char *sName) = 0;
    virtual void AddMobileObj(AbstractMobileObj *pMobObj) = 0;
};
```

4.4 Klasa abstrakcyjna `AbstractComChannel`

Klasa zawiera niezbędne metody do realizacji komunikacji z serwerem.

```
class AbstractComChannel {
 public:
  /*!
   * \brief Inicjalizuje destryptor gniazda.
   *
   * Inicjalizuje destryptora pliku skojarzonego z połączeniem sieciowym z serwerem.
   * \param[in] Socket - zawiera poprawny deskryptor.
   */
```

```
    virtual Init(int Socket) = 0;
   /*!
    * \brief Udostępnia deskryptor pliku skojarzonego z połączeniem sieciowym z serwerem.
    *
    *  Udostępnia deskryptor skojarzonego z połączeniem sieciowym z serwerem.
    * \return Deskryptor pliku.
    */
    virtual int GetSocket() const = 0;
   /*!
    * \brief Zamyka dostęp gniazda.
    */
    virtual void LockAccess() = 0;
   /*!
    * \brief Otwiera dostęp do gniazda.
    */
    virtual void UnlockAccess() = 0;
    /*!
     * \brief Udostępnia mutex w trybie modyfikacji.
     *
     *  Udostępnia mutex w trybie modyfikacji.
     *  Jest to przydatne, gdy planowany jest inny typ zamknięcie,
     *  np. poprzez klasę std::lock_gaurd, która daje możliwość
     *  bezpieczniejszego zamknięcia.
     */
    virtual std::mutex &UseGuard();
};
```

5 Interfejs wtyczek

Każda wtyczka realizująca pojedynczą komendę powinna udostępniać następujące funkcje:

`const char* GetCmdName()` – zwraca wskaźnik do napisu będącego nazwą danego polecenia. W przypadku polecenia `Move` będzie to wskaźnik na napis ''`Move`''.

`Interp4Command* CreateCmd()` – tworzy obiekt modelujący dane polecenie.