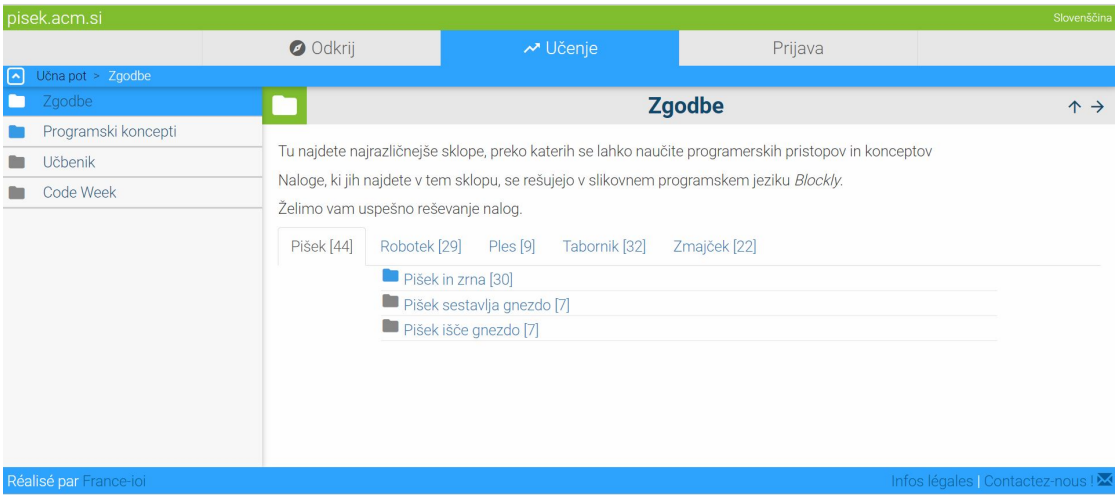
Navodila za uporabo portala Pišek

Špenko Krištof, Flajs Žiga, Dobravec Blaž

June 28, 2019

Contents

1 Osnovno



Portal je v osnovi razdeljen na različne sklope:

- Zgodbe
- Programski koncepti
- Učbenik
- Code Week

V vsakem od sklopov so različne naloge, vendar **POZOR** v sklopu *Zgodbe* in sklopu *Učbenik* so naloge originalne, v **vseh** ostalih sklopih pa so naloge le kopije teh iz Učbenika in Zgodb.

V zavihku Odkrij oz. Navodila se nahaja text, kjer je opisano kako je nastal portal Pišek in podrobnosti, kontakt/ financiranje ipd.
To se ureja na SVN na naslovu:

..\Slovenia\Ostalo\Navodila\index.html

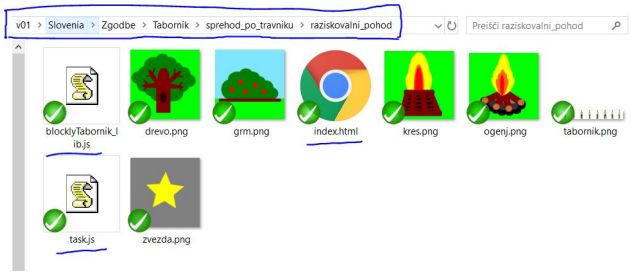
2 Izdelava nove naloge

2.1 Splošna navodila

Ta del se nanaša na vse tipe nalog, torej so neka splošna navodila, kjer se obravnava stvari, ki so skupne vsem nalogam. razloži se tudi neke splošno razdelavo posamezne datoteke.

V mapi na SVN repozitoriju, kjer se nahaja naloga, so naslednje datoteke:

- *index.html*
- *taks.js*
- Knjižnica z lastnostmi posamezne datoteke (ta del je začasen, saj bi bilo v prihodnosti bolje, da so te datoteke definirane samo na enem mestu.), ki so pomenovane: *blocklyRobot_lib.js* ali *blocklyPrinter_lib.js* ali *blocklyTurtle_lib.js* ta datoteka je ustrezno poimenovana, v primeru da smo v zgodbi Tabornik, se tudi datoteka imenuje: *blocklyTabornik_lib.js*
- v primeru, da ima naloga vključene fotografije, se le te prav tako nahajajo tukaj



2.1.1 index

Ta datoteka se uporablja predvsem za to, da postavimo ogrodje samega izgleda, oziroma, da zloži vse komponente v neko celoto.

V tej datoteki se napiše tudi besedilo vsake od podnalog. Obenem pa se znotraj te datoteke vključi vse potrebne module, prav tako lahko tu vključimo kakšne nove module, ki bodo dostopni v prihodnosti.

Nastavi se tudi jezik in nekatere spremenljivke.

```
<!doctype html>
<html>
  <head>
   <title>Taborniško državno prvenstvo</title>
   <meta charset="utf-8">
    <script>
      window.stringsLanguage = 'sl';
    </script>
    <script class="remove" type="text/javascript" src="../../../../../_common/modules/pemFioi/importModules-1.1.js" id="import-modules"><
    <script class="remove" type="text/javascript">
      var modulesPath = '../../../../../_common/modules/';
      importModules([
          'jquery-1.7.1', 'JSON-js', 'raphael-2.2.1', 'beaver-task-2.0', 'jschannel', 'raphaelFactory-1.0',
          'delayFactory-1.0', 'simulationFactory-1.0',
          'platform-pr', 'buttonsAndMessages', 'beav-1.0', 'installationAPI.01', 'miniPlatform',
          'taskStyles-0.1', 'jwinf_css']);
      // set Blockly as default language when none is specified through ?language=
      importLanguageModules('blockly');
    </script>
    <script class="remove" type="text/javascript" src="blocklyPrinter_lib.js" id="blocklyPrinter_lib"></script>
    <script class="remove" type="text/javascript">
```

Na začetku imamo naslov naloge (to ni dejansko naslov naloge temveč le ime, ki bo pisalo v zavihku v brskalniku), nato določimo jezik samega okna. V naslednjih vrsticah najprej uvozimo modul: *importModules-1.1.js* oziroma katerokoli višjo/nižjo različico modulov. To je potrebno za samo lokalno testiranje delovanja našega programa.

Pri uvažanju modula importModules-1.0 naloga dela na strežniku, vendar ne upošteva lokalne knjižnice blocklyRobot_lib.js. Če pa se uporabi imporModules-1.1. naloga upošteva lokalno knjižnico, vendar ne dela na strežniku.

Zelo pomembno pri teh naslovih je kako nizko v drevesni strukturi se nahajamo, saj je od tega odvisno število "../" pred samim naslovom, prav tako je potrebno to, da na enak način spremenimo spremenljivko modulesPath, v naslednji vrstici.

Če teh nastavitev ne nastavimo pravilno se nam na lokalno naloga in sam index.html ne bo pravilno izpisal. Nato uvozimo še datoteko textitblocklyPrinter_lib.js oziroma ustrezno poimenovano datoteko glede na tip naloge in samega junaka.

```
<script class="remove" type="text/javascript">
var json = {
    "id": "naloge-ucbenik-34-1",
    "language": "sl",
    "version": "sl.01",
    "authors": "Luka Čušin",
    "translators": [],
    "license": "",
    "taskPathPrefix": "",
    "modulesPathPrefix": "",
    "browserSupport": [],
    "acceptedAnswers": [""],
    "fullFeedback": true,
    "minWidth": "auto"
};
var strings = {
    moreThan100Moves: "Naredil/-a si več kot 100 potez.",
    notAllCellsPainted: "Vsa polja nisi pobarval/-a pravilno. Poskusi še enkrat!",
    allCellsPainted: "Čestitamo! Vsa polja si pravilno pobarval/-a!"
};
</script>
<script type="text/javascript" src="printer_task.js"></script>
```

V spremenljivko *json* shranimo podatke, kot so avtor naloge, jezik naloge in nekatere druge podatke, kot so *fullFeedback* ipd.
V spremenljivko *strings* shranimo nekatere splošne podatke.

Dodamo fotografije, ki jih bomo uporabljali:

```
        "browserSupport": [],
        "fullFeedback": true,
        "minWidth": "auto"
    };
</script>


<img src="green_robot.png" style="display:none" />

<img src="obstacle.png" style="display:none" />
<img src="green.png" style="display:none" />

<img src="marble.png" style="display:none" />
<img src="hole.png" style="display:none" />
```

Na koncu samega *head*-a pa uvozimo še ustrezno javascript datoteko, katere ime je prav tako odvisno od tipa naloge.

Če želimo dodati **namig** v nalogo, ga dodamo na tem mestu:

```
</script>
<script type="text/javascript" src="printer_task.js"></script>
 <script>
    function hint(id){
        var hint = document.getElementById(id);
        if (hint.style.display == "none") {
            hint.style.display = "block";
        }
        else {
            hint.style.display = "none";
        }
    };
 </script>
```

Tu pa se začne dejansko besedilo vsake od nalog:

```
<body onresize="task.displayedSubTask.updateScale()">
    <div id="task">
      <h1>Taborniško državno prvenstvo</h1>
     <div id="tabsContainer"></div>
     <div id="taskContent">
       <div id="taskIntro">
        <p class="easy">
          Tabornik Tine je dobil častno nalogo!
          Razglasiti mora rezultate tekmovanja Preživetje v naravi.
          Za pripravo govora ni dobil dovolj časa, zato si je nekaj
          uvodnih besed na hitro zapisal na list.
          Besede, zapisane v vrsticah, združi v celoto, da jih bo Tine lažje prebral.
        </p>
        <p class="medium">
          Tinetu je uspelo pravočasno zapisati govor, a
          ga je pred razglasitvijo rezultatov še enkrat pregledal.
          Nekatere besede v njem mu niso bile všeč in jih je označil z <code>#</code>,
          da jih med govorom izpusti. Sestavi popravljen govor za Tineta.
        </p>
        <p class="hard">
           Tine še vedno popravlja govor. Z <code>#</code> je označil še nekaj neuporabnih besed. Pre
           da se bo med govorom spomnil narediti premor. Da bo premorelažje videl, zapiši vrstice pre
        </p>
       </div>
       <div id="gridContainer"></div>
       <div id="blocklyLibContent"></div>
      </div><!-- task -->
 </body>
</html>
```

V tem delu definiramo kakšnega tipa je naloga, poznamo naslednje težavnosti:

- easy
- medium
- hard

Od tega je tudi odvisno koliko zvezdic se bo pokazalo v zgornjem zavihku.
Znotraj teh odstavkov pa stavimo dejansko navodilo naloge, ki je potem vidno uporabniku.
V primeru da teh tipov ne definiramo a vseeno imamo več podnalog, se besedilo prikaže pri vseh podnalogah, kot kaže spodnja fotografija.

```
<div id="taskContent">
    <div id="taskIntro">
    <p>
        Tabornik Tine potrebuje še več vaje, da bo orient
        Z drugimi taborniki, se je zato podal na taborniš
        Popravi program tako, da bo Tine hodil po pravi p
    </p>
   </div>
    <div id="gridContainer"></div>
    <div id="blocklyLibContent"></div>
</div>
```

2.1.2 task

V tej datoteki, nastavimo parametre samega dela naloge, v kodo vstavimo že obstoječe bloke, nastavimo omejitve števila blokov, ustvarimo teste, ki bodo preverjali pravilnost našega programa, ipd.

Koda se začne z definicijo same funkcije:

```
function initTask(subTask) {
```

Nato se začne del, v katerem določimo lastnosti samega Blockly okolja.

```
subTask.gridInfos = {
   hideSaveOrLoad: false,
   actionDelay: 200,
   includeBlocks: {
      groupByCategory: true,
      generatedBlocks: {
          printer: ["print", "read"],
      },
       // Block math_floor_divison not avalible
      standardBlocks: {
         includeAll: false,
         wholeCategories: ["variables"],
         singleBlocks: { easy : ["text", "text_join", "controls_repea
                         medium : ["text", "text_join", "text_append"
                         hard : ["text", "text_join", "text_append",
                       },
      }
   },
```

Tu določimo naslednje lastnosti:

- hideSaveOrLoad: false → kar pomeni, da se bo prikazal gumb, kjer lahko shranimo kodo ali jo kopiramo.

- **actionDelay: 200** → ki ob kliku za določen čas v milisekundah ustavi delovanje funkcije
- **includeBlocks:** → v katerem definiramo, da so bloki grupirani v skupine, ter katere bloke zgenerira.
  Tukaj je seznam kratic za vse standardne bloke katerih kratice je potrebno poznati, ce želimo v nalogo dodati eno specificno kocko:

  https://gist.github.com/davidferguson/f85e1ae84b940bf8398f9bdbe98a4d8f

- **standardBlocks:** → v katerem lahko nastavimo, da ne vključi vseh blokov, nastavimo, da je vidna celotna skupina ipd.
- **maxInstructions: (easy: 0, medium: 0, hard: 10)** → kar pomeni število dovoljenih blokov, ki jih uporabnik lahko uporabi. v primeru, da je število blokov 0, to pomeni, da omejitev števila blokov ni.
- **checkEndEveryTurn: false** → kar pomeni, da je preverjanje na vsakem koraku izklopljeno, kar pomeni, da preverja samo končno situacijo.
- **blocklyColourTheme: "bwinf"** → Pomeni, da uporablja eno od blockly-ijevih tem.
- **checkEndCondition: function(context, lastTurn)** → je klic funkcije, ki preverja pravilnost programa → ta funkcija se nahaja v *blocklyTabornik_lib.js* oziroma v korespondentno poimenovani datoteki glede na samo tematiko.
  V tem delu lahko definiramo popolnoma svoje teste.
- **computeGrade: function(context, message)** → je funkcija, ki izračuna oceno kakšen procent rešitve je uporabnik pravilno naredil.

```
,,
computeGrade: function(context, message) {
    var rate = 0;
    if (context.success) {
        rate = 1;
        if (context.nbMoves > 100) {
            rate /= 2;
            message += languageStrings.messages.moreThan100Moves;
        }
    }
    return {
        successRate: rate,
        message: message
    };
}
```

Nato pridemo do naslednjega dela, ki ga imenujemo *data*, pri katerem napišemo lastnosti posameznih težavnosti. Ta del je razdeljen enako kot so razdeljena navodila v pripadajočem *index.html*.

```
  subTask.data = {
      easy: [
        {
          input: "Pozdravljeni,\ntaborniki.\nMoje\nime\nje\nTine.",
          output: "Pozdravljeni, taborniki. Moje ime je Tine.",
        },
      ],
      medium: [
        {
          input: "Danes ste se na\n#taborniškem\ntekmovanju\n#vsi\nodlično izkazali.",
          output: "Danes ste se na tekmovanju odlično izkazali.",
        },
      ],
      hard : [
         {
            input: "Čas za razglasitev\n#sledečih\nrezultatov:\n%tretje mesto\n#aplavz\n
            output: "Čas za razglasitev rezultatov:\ntretje mesto je osvojila taborniška
         },
      ],
  };

  initBlocklySubTask(subTask);
}

initWrapper(initTask, ["easy", "medium", "hard"], null);
```

V primeru, da želimo postaviti več različnih testov, lahko znotraj posamezne težavnosti (torej, easy/medium/hard), podamo več različnih testov, kot kaže spodnja fotografija.

```
easy: [
  {
    tiles: [
        [3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
        [3, 3, 2, 4, 1, 1, 1, 1, 4, 2, 9, 1, 1, 4, 2],
        [3, 2, 2, 1, 2, 2, 4, 1, 1, 1, 1, 4, 2, 1, 2],
        [2, 2, 2, 1, 2, 2, 4, 1, 1, 1, 1, 1, 1, 4, 2],
        [2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2],
        [2, 2, 2, 4, 1, 1, 1, 1, 1, 1, 1, 1, 4, 2, 3, 3],
        [2, 1, 1, 1, 1, 1, 1, 1, 1, 4, 2, 2, 2, 2, 3, 3],
        [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
      ],
    initItems: [
        { row: 6, col: 1, dir: 0, type: "green_robot" }
      ]
  },
  {
    tiles: [
        [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
        [2, 4, 1, 4, 2, 2, 2, 2, 9, 1, 1, 1, 1, 4, 2],
        [2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2],
        [2, 1, 2, 1, 2, 4, 4, 2, 3, 3, 3, 3, 2, 1, 2],
        [2, 4, 1, 1, 1, 1, 4, 2, 3, 3, 3, 3, 2, 1, 2],
        [2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2],
        [2, 1, 1, 4, 2, 4, 1, 1, 1, 1, 1, 1, 1, 4, 2],
        [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
      ],
    initItems: [
        { row: 6, col: 1, dir: 0, type: "green_robot" }
      ]
  },
```

Na koncu v oglatih oklepajih v *initWrapper* dodamo to funkcijo initTask, težavnosti, ki jih imamo, ter *null*.

2.2 Mreža

To so naloge, pri katerih se junak/več junakov premika po mreži.

2.2.1 task

- **cellSide** → ki nastavi velikost enega kvadratka mreže (oziroma velikost mreže)
- **backgroundColor** → spremenljivka določa barvo polj (barva zapisana v HEX)
- **borderColor** → spremenljivka določa barvo roba polj (barva zapisana v HEX)
- **noBorder** → izberemo ali črte med kvadratki narišemo

```
actionDelay: 200,
itemTypes: {
    tabornik: { img: "tabornik.png", category: "robot", side: 80, nbStates: 9, offsetX: -15 },
    obstacle1: { num: 2, img: "grm.png", side: cellSide, isObstacle: true },
    obstacle2: { num: 5, img: "drevo.png", side: cellSide, isObstacle: true },
    green: { num: 3, img: "cilj.png", side: cellSide, color: "vert" },
    pill1: { num: 4, img: "zvezda.png", side: cellSide, category: "pill", isObstacle: false, isCollectible: true, zOrder:
},
maxInstructions: 0,
```

Te tipi imajo naslednje lastnosti:
(odvisno je tudi od tega kakšnega tipa je objekt)

- **img** → ki se navezuje na fotografijo, ki je bila definirana v index-u ter se nahaja na istem direktoriju kot ta *javascript* datoteka.
- **category** → ki se nanaša na to katere kategorije je objekt
- **num** → ki definira številko objekta, ki jo potem uporabi pri testih.
- **side** → ki pove kakšna je velikost kvadratka.
- **isObsticle** → ki pove če je objekt ovira ali ne
- **isCollectable** → ki pove, če se objekt lahko pobere
- **zOrder** → ki nam pove orientacijo v prostoru, kateri predmet bo pred drugim
- **nbStates** → število zazličnih pozicij, ki jih lahko ta objekt ima, torej vključuje rotacije ipd.
- **offsetX** → ki zamakne objekt v x-osi.

Obstajajo tudi drugi parametri, ki si jih lahko pogledate tukaj: *https://github.com/France-ioi/bebras-modules/tree/master/pemFioi/quickAlgo*.

Pri tipu naloge *mreža* je *data* definiran tako, da definiramo:

```
subTask.data = {
   easy: [
      {
         tiles: [
            [1, 4, 4, 4, 4, 4],
            [2, 4, 2, 2, 2, 4],
            [2, 4, 2, 2, 2, 4],
            [2, 4, 4, 4, 4, 4]
         ],
         initItems: [
               { row: 0, col: 0, dir: 0, type: "tabornik" }
            ]
      }
   ],
   medium: [
      {
```

- **tiles** → ki predstavlja tabelo številke v njej pa se nanašajo na številke fotografij definirane zgoraj v tipih objektov, številka 1 pomeni da je polje prazno **(na tem mestu ne definiramo premikajočih se elementov)**.
- **initItems** → kjer nastavimo zečetno pozicijo našega/naših junakov, v primeru da jih je več, jih naštejemo več.

```
            [1, 1, 1, 1, 1, 1, 1],
            [1, 1, 1, 1, 1, 1, 5],
            [1, 1, 1, 1, 1, 1, 1],
            [1, 1, 1, 1, 1, 5, 2]
         ],
         initItems: [
            { row: 0, col: 2, dir: 2, type: "plesalec3" },
            { row: 1, col: 0, dir: 0, type: "plesalec2" },
            { row: 3, col: 0, dir: 0, type: "plesalec1" },
            { row: 5, col: 0, dir: 0, type: "plesalec"  },
            { row: 0, col: 0, dir: 0, type: "plesalka"  },
         ]
      },
```

2.3 Risanje

Pri tem tipu naloge gre za risanje nekega junaka na platno.

2.3.1 task

najprej v *generatedBlocks* vstavimo definirane bloke.

```
generatedBlocks: {
   easy: {turtle: ["moveamount", "turnleftamount", "turnrightamount"]},
   medium: {turtle: ["moveamount", "turnleftamount", "turnrightamount"]},
   hard: {turtle: ["moveamount", "movebackamount", "turnleftamount", "turnrightamount"]},
},
standardBlocks: {
```

Nastavimo korak junaka:

```
overlayFileName: 'sand.png',
turtleStepSize: 0.1,
coords: {x: 240, y: 200},
maxInstructions: {easy: 0, medium: 0, hard: 0},
```

Lastnost *coords* premakne celotno fotografijo.

V Preverjanje pravilnosti programe torej v *checkEndCondition* napišemo funkcijo, ki jo definiramo v ali kar v *task.js* ali pa v korespondenčni knjižnici, v našem primeru v *blocklyTurtle_lib*.

```
checkEndCondition: function(context, lastTurn) {
  if (lastTurn) {
    var userImage = context.turtle.invisibleTurtle.drawingContext.getImageData(0, 0, 300, 300);
    var solutionImage = context.turtle.invisibleSolutionTurtle.drawingContext.getImageData(0, 0, 300, 300
    var len = Math.min(userImage.data.length, solutionImage.data.length);
    var delta = 0;
    var fill = 0;
    var empty = 0;
    // Pixels are in RGBA format.  Only check the Alpha bytes.
    for (var i = 3; i < len; i += 4) {
      // Check the Alpha byte.
      if (Math.abs(userImage.data[i] - solutionImage.data[i]) > 127) {
        delta++;
      }
      if (solutionImage.data[i] > 127)
        fill++;
      else
        empty++;
    }
```

Data je lahko definiran tako:

```
subTask.data = {
  easy: [{
    drawSolution : function(turtle) {
      for (var i = 0; i < 6; ++i) {
        turtle.move(100);
        turtle.turn(60);
      }
    },
  }],
  medium: [{
    drawSolution : function(turtle) {
      for (var i = 0; i < 6; ++i) {
        turtle.move(100);
        turtle.turn(60);
      }
    },
  }],
  hard: [{
    drawSolution : function(turtle) {
      for (var i = 0; i < 6; ++i) {
        turtle.move(100);
        turtle.turn(60);
      }
    },
  }],
};
```

Kjer v premenljivko *drawSolution* shranimo pravilno rešitev.

2.4 Vhod/izhod

To so naloge, pri katerih je potrebno napisati nekaj v okno. ta rezultat se preveri.

2.4.1 task

Naloge delujejo tako, da skozi spremenljivki *input* in *output* dodamo kako naj bi izgledala naloga in kaj bi bila pravilna rešitev.

```
subTask.data = {
    easy: [
      {
        input: "Pozdravljeni,\ntaborniki.\nMoje\nime\nje\nTine.",
        output: "Pozdravljeni, taborniki. Moje ime je Tine.",
      },
    ],
    medium: [
      {
        input: "Danes ste se na\n#taborniškem\ntekmovanju\n#vsi\nodlično izkazali.",
        output: "Danes ste se na tekmovanju odlično izkazali.",
      },
    ],
    hard : [
        {
            input: "Čas za razglasitev\n#sledečih\nrezultatov:\n%tretje mesto\n#aplavz\nje osvojil
            output: "Čas za razglasitev rezultatov:\ntretje mesto je osvojila taborniška skupina O
        },
    ],
};
```

2.5 Specifični sklopi

2.5.1 Tabornik

- Naloge sklopa Tabornik delujejo z modulom importModules-1.1., vendar samo ce se lokalna knjižnica nahaja v isti datoteki kot index.
-

3 Administratorski vmesnik

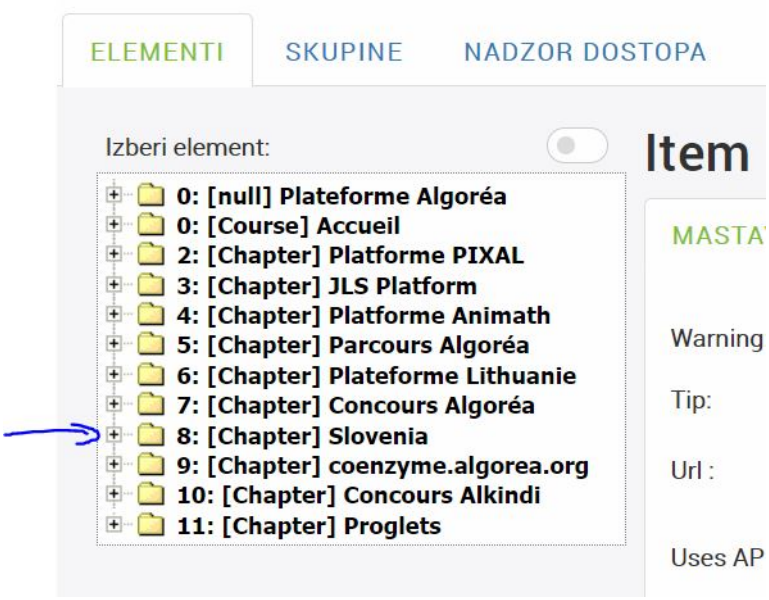
Administratorski vmesnik, se uporablja predvsem zato, da upravljalci nalog lahko konkretno testirajo, in upravljajo z objavljenimi nalogami. Najprej je potrebno navigirati na:

http://pisek.acm.si/admin/index.html

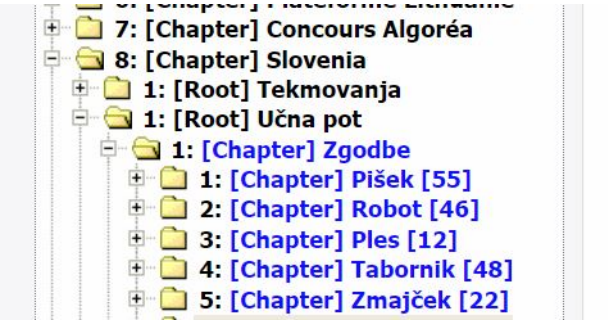
Nato vpišete naslednje uporabniško ime in geslo:

Uporabniško ime: pronal
geslo: pronal_pisek

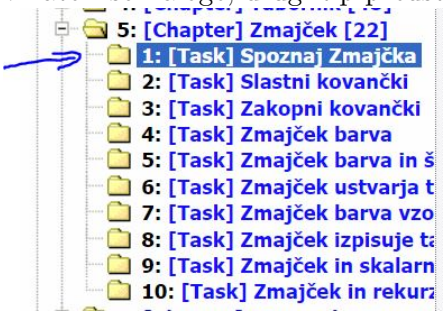
Ko ste prijavljeni, lahko na levi strani vidite drevesno strukturo, na desni strani pa so različni zavihki, v katerih lahko urejate različne sklope/naloge ipd.



Naloge se nahajajo v mapi slovenija, ter znotraj podmape *Učna pot*.

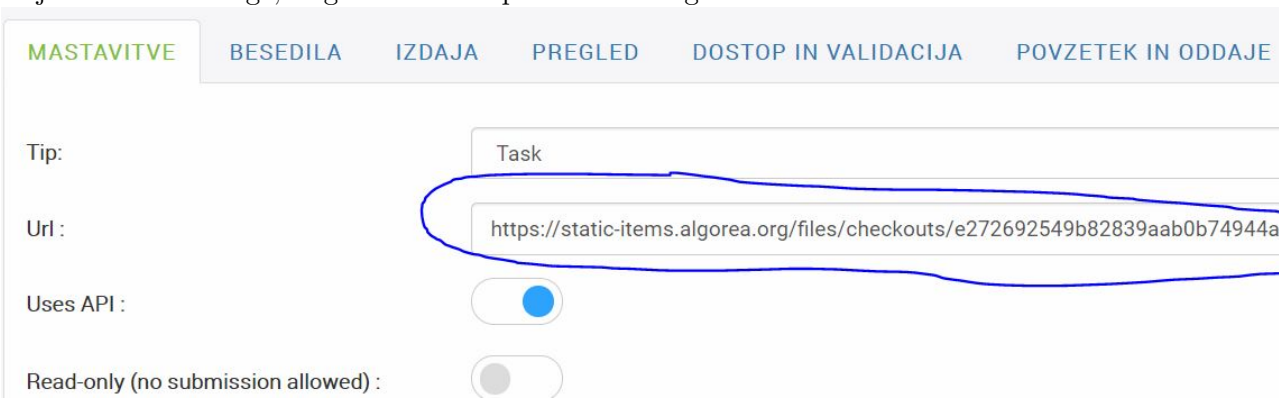

7: [Chapter] Concours Algoréa
8: [Chapter] Slovenia
1: [Root] Tekmovanja
1: [Root] Učna pot
1: [Chapter] Zgodbe
1: [Chapter] Pišek [55]
2: [Chapter] Robot [46]
3: [Chapter] Ples [12]
4: [Chapter] Tabornik [48]
5: [Chapter] Zmajček [22]

Znotraj Piška poznamo dva tipa, tip *Chapter* in tip *Task*. Prvi tip predstavlja mapo, v kateri so naloge, drugi tip predstavlja dejansko nalogo.

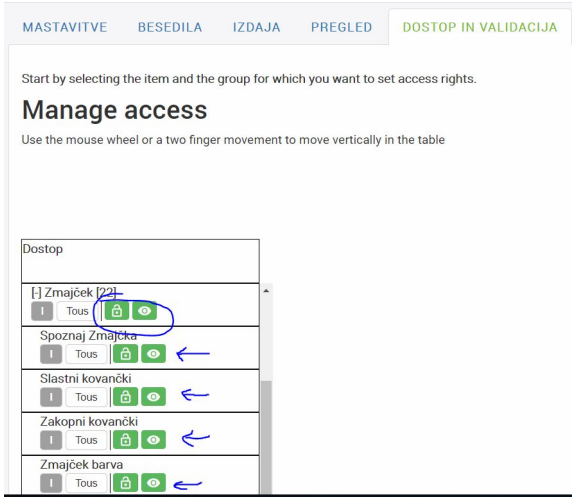


Na desni strani, so v zavihkih različne funkcije, ki se nanašajo predvsem na lastnosti naloge oziroma sklopa, torej:

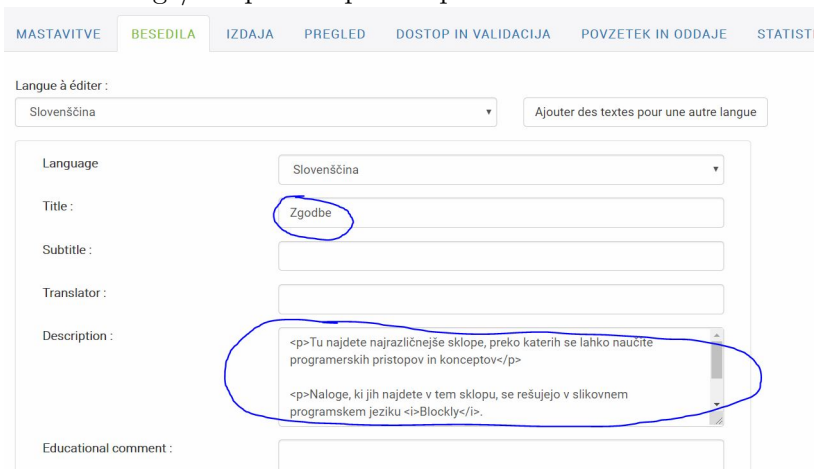
- Dejanski URL naloge, ki ga nastavimo pri uvozu naloge.



- Validacija naloge, ki nalogo naredi skrito oziroma odprto.



- Naslov naloge/sklopa ter opis sklopa



V sami drevesni strukturi, lahko naloge/sklope tudi kopiramo ali jih izbrišemo (desni klik), vendar moramo biti pri tem pozorni, saj je potrebno ločevati med tem da nalogo naložimo z drugačnim URL-jem na portal, ker se v primeru kopiranja, dejanski url prav tako kopira, kar pomeni, da če nalogo spremenimo in jo posodobimo na portalu, se bo naloga spremenila tudi na klonu naloge. V primeru da original izbrišemo, se klon ne izbriše.

Podobne modifikacije se da delati tudi na dejanskem Portalu Pišek, le z nekaterimi omejitvami. Prijavno okno je bilo skrito (s strani Administratorjev strežnika - Za spremembo je potrebno kontaktirati le te)

povezava za samo prijavo pa se nahaja na tem naslovu:

https://login.france-ioi.org/auth

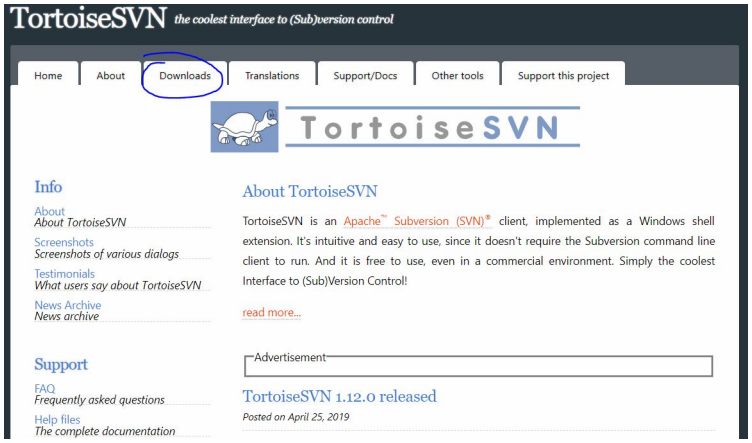
Geslo in uporabniško ime sta enaka, kot za administratorski vmesnik. V samem administratorskem vmesniku so še nekatere dodatne funkcije/lastnosti nalog in sklopov, ki pa si jih poglejte sami.

4 SVN Repozitorij

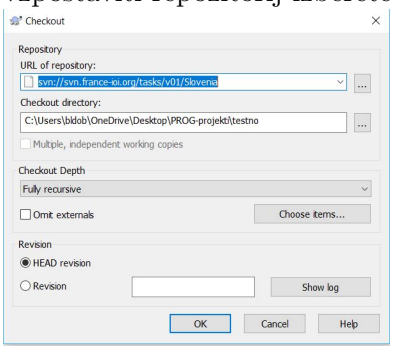
4.1 Vzpostavitev SVN repozitorija

Za uspešno vzpostavitev repozitorija je potrebno imeti program, ki podpira SVN-repozitorija. V našem primeru bomo uporabili **TurtioseSVN**. Program lahko dobite na tem naslovu:

https://tortoisesvn.net/downloads.html



Ko je program naložen, lahko s pomočjo desnega klika na lokacijo, na katero želite vzpostaviti repozitorij izberete možnost *SVN Checkout*.



v URL of repository vpišemo naslenje:

svn://svn.france-ioi.org/tasks/

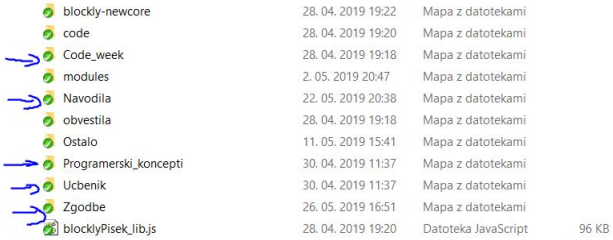
Ko kliknemo Ok, se prične nalaganje, ko se naložijo vse datoteke, ste vzpostavili SVN-repozitorij.

4.2 Struktura SVN repozitorija

SVN je strukturiran tkao, da imamo najprej zunanjo mapo, ki je podobna kot pri administratorskem vmesniku, nato pa podobno kot tam navigiramo v mapo Slovenia.

Kar se tiče drevesne strukture na SVN vmesniku, je tako, da je struktura postavljena kar se da podobno kot struktura na samem portalu.
Natančneje:

Sklopa *Zgodbe* in sklop *Učbenik* sta sklopa, v katerih se nahajajo originalne naloge, medtem ko v sklopih *Navodila*, kjer se nahajajo ta navodila in vse datoteke za spreminjanje, navodil, *CodeWeek*, kjer načeloma ni ničesar, razen, če se v prihodnosti na CodeWeek objavljajo originalne naloge, Na portalu, pa so večinoma naloge kopije nalog iz zgodb in učbenika. Enako velja za sklop *Programski koncepti*. Kar se tiče ostalih map, so to mape, ki vključujejo različne module, ki so potrebni za uspešno izvajanje in lokalno testiranje.

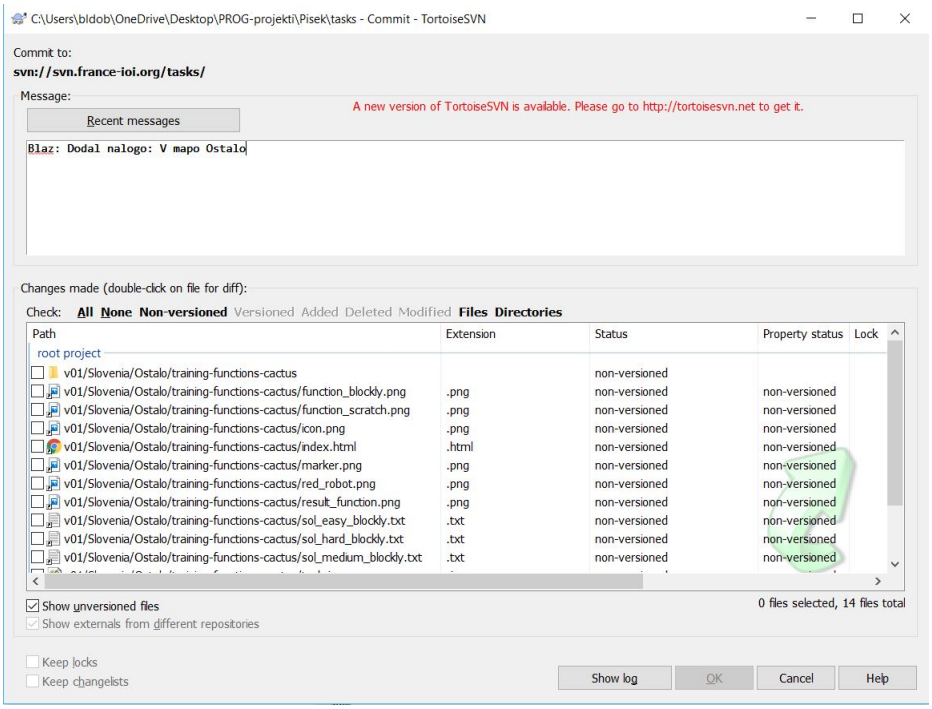


Vsebina posamezne naloge in pot izgleda približno takole:

Tudi pot je takšna, kot je pot na dejanskem portalu. To sicer pomeni več dela v primeru premikanja nalog, vendar se v večini od teh nalog nebi smela spreminjati naslov/pot. Kot je bilo opisano v sklopu Nove naloge so pomembne podčrtane datoteke.

4.3 Nalaganje na SVN repozitorij

Naloge najlažje dodamo na SVN tako, da jih dejansko ustvarimo (oziroma kopiramo že obstoječo nalogo in jo preuredimo, tako, kot si jo želimo). Nato pa z desnim gumbom kliknemo na lokacijo SVN-repozitorija in izberemo možnost *SVN Commit...*
Prikaže se nam tako okno:



kjer napišemo sporočilo v obliki:

IME: Sporočilo

Izberemo datoteke, ki jih želimo naložiti na repozitorija in kliknemo OK. Naloga se naloži.

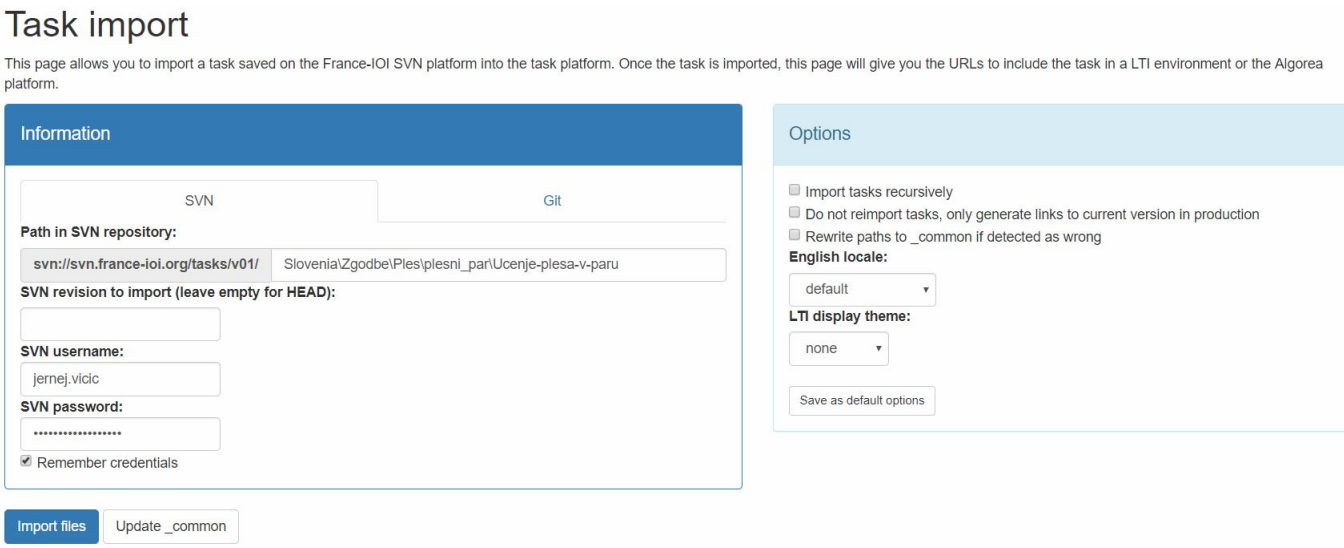
5 Nalaganje na Portal Pišek

Ko imamo nalogo na SVN repozitoriju, jo lahko impotiramo tudi na portal, to naredimo prek naslednje povezave:

http://svnimport.france-ioi.org

Potrebno je vpisati uporbniško ime in geslo:

Uporabniško ime: jernej.vicic
geslo: slov798xb2127x9ssX



Na ustrezna mesta vpišemo podatke, dodamo pot v SVN repozitoriju, kot je prikazano na zgodnji sliki.

Ko se naloga uvozi, se pojavi povezava, s katero dostopate do lokacije naloge.

Če na SVN-ju ne spreminjamo drevesne strukture (in ne preimenujemo map) in posodabljamo datoteke (commitamo) in nalogo naložimo na strežnik,

je naceloma dovolj samo osvežiti piškovo spletno stran, pa bi morala biti opazna razlika. V primeru da se to ne zgodi, poskusite zbrisati zgodovino podatkov v vašem brskalniku.

Nato v novem zavihku navigirate na administratorski vmesnik, ki se nahaja tukaj:

http://pisek.acm.si/admin/index.html

Kliknite na ustrezno željeno mesto v drevesni stukturi z desnim klikom se vam pokažejo možnosti, izberite možnost new. Na tak način ustvarite enoto. Če izberete tip "Chapter" ustvarite imenik, če pa "Task" pa nalogo. Tako lahko ustvarimo poljubno drevesno strukturo in vanjo uvrstimo naloge.

Za nalogo je dovolj nastaviti le URL, pravilen tip in vklopiti "Uses API" (ki je privzeto vklopljen).